ADAPTIVE DIALOGUE SYSTEMS FOR ASSISTIVE LIVING ENVIRONMENTS

by

ALEXANDROS PAPANGELIS

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington, in the context of the joint PhD programme
with the National Center for Scientific Research “Demokritos”, in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
December 2013

Copyright © by ALEXANDROS PAPANGELIS 2013

All Rights Reserved

To my parents, Eleni and Vassilis, my sister Irinna and brother Nasos and to my soulmate Myrna, for their continuous love and support throughout these hard but rewarding years.

ACKNOWLEDGEMENTS

I would like to thank my supervising professors Dr. Fillia Makedon and Dr. Vangelis Karkaletsis, for their guidance, patience and support. I would also like to thank my Ph.D. Committee Members, Dr. Heng Huang, Dr. Constantine Spyropoulos and Dr. Chris Ding. Last, I would also like to thank my colleagues and collaborators, without whose a great portion of this thesis would not have been possible.

November 15, 2013

ABSTRACT

ADAPTIVE DIALOGUE SYSTEMS FOR ASSISTIVE LIVING ENVIRONMENTS

ALEXANDROS PAPANGELIS, Ph.D.

The University of Texas at Arlington, 2013

Supervising Professor: Fillia Makedon

Adaptive Dialogue Systems (ADS) are intelligent systems, able to interact with users via multiple modalities, such as speech, gestures, facial expressions and others. Such systems are able to make conversation with their users, usually on a specific, narrow topic. Assistive Living Environments are environments where the users are by definition not competent with technology, due to various factors, such as mental or physical disabilities, age and others. While technology that helps improve these people's quality of life exists, many times they cannot access it due to inflexible interfaces. ADS, therefore, have the potential to bridge users and technology by acting as a mediator between them. There are several unique challenges posed by this problem, in addition to the challenges faced by a generic ADS. Our contributions to the state of the art focus on Online Dialogue Policy learning which, coupled with other methods we proposed, can lead to an ADS able to exhibit complex behaviour and appear more intelligent. As a consequence, users trust the system more and it becomes more functional as it is able to elicit behavioural information and use it, for example, to make basic diagnoses.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iv

ABSTRACT v

LIST OF FIGURES x

LIST OF TABLES xiv

Chapter Page

1. Introduction 1
    - 1.1 Recent Advances in Assistive Living Environments 3
        - 1.1.1 Users' Perception of ALE 4
        - 1.1.2 Assistive Technology 7
        - 1.1.3 Other Approaches, Frameworks and Systems 9
    - 1.2 Recent Advances in Adaptive Dialogue Systems 10
        - 1.2.1 Adaptive Dialogue Systems 12
        - 1.2.2 Dialogue Management 24
        - 1.2.3 User Simulation 39
        - 1.2.4 Evaluation 42
    - 1.3 Discussion 47
2. Assistive Living Technologies 49
    - 2.1 Sensor Placement in Assistive Living Environments 49
    - 2.2 Max-Sum Decentralised Coordination Algorithm 50
    - 2.3 The Extended Max-Sum Decentralised Coordination Algorithm . . . 52
        - 2.3.1 Experimental Setup 54
        - 2.3.2 Results 55

2.3.3 System response to environmental changes 56
2.3.4 Fault Tolerance 57
2.4 EMSDC Web Tool 59
2.4.1 Environment Design 61
2.4.2 System Monitoring 63
2.4.3 Evaluation 63
2.5 Recommender System for Assistive Living Environments 65
2.5.1 Introduction to Recommender Systems 66
2.5.2 Related Work 68
2.5.3 A Recommender System for ALE 69
2.5.4 Conclusions 76
2.6 Discussion 77
3. Online Dialogue Policy Learning 78
3.1 Background Knowledge 78
3.1.1 Reinforcement Learning 79
3.1.2 Modeling the Dialogue Problem 84
3.1.3 Information Seeking Paradigm 86
3.2 Evaluation of Reinforcement Learning Methods 90
3.2.1 Experimental Setup 93
3.2.2 User Simulator 93
3.2.3 Results 93
3.3 Discussion 95
4. Towards Complex Behaving Adaptive Dialogue Systems 96
4.1 Motivation 96
4.2 Related Work 98
4.2.1 Hierarchical Reinforcement Learning 101

4.2.2 Complex Actions 102

4.3 Complex action learning 103

4.4 Goal Achievement 106

4.5 Evaluation 112

4.5.1 User simulator 113

4.5.2 Experimental setup 116

4.5.3 Results 119

4.6 Impact of AWL and USE on the interaction 127

4.6.1 Navigation ADS 127

4.6.2 Object Manipulation ADS 131

4.7 Discussion 134

5. Applications of Adaptive Dialogue Systems in Assistive Living Environments 138

5.1 Museum Guide 138

5.1.1 System Overview 139

5.1.2 Evaluation 143

5.2 Navigation 146

5.3 Object Manipulation 146

5.3.1 System Overview 147

5.3.2 Adaptive Dialogue System 149

5.3.3 Discussion 153

5.4 Adaptive Dialogue Management Development Framework 154

5.4.1 State transitions or updates 156

5.4.2 Adaptive Dialogue Management 157

5.4.3 Learning dialogue strategies 158

5.4.4 Transformation to Dialogue Models 159

5.5 Discussion 162

6. Assessing Post Traumatic Stress Disorder 164

6.1 Post Traumatic Stress Disorder 165

6.2 Modelling the Interaction 172

6.3 PTSD Assessment ADS Overview 178

6.4 Usability Inspection and Evaluation 186

6.5 Evaluation 188

6.6 Discussion 189

7. Discussion 191

7.1 Open Problems 191

7.2 Future Directions 192

7.2.1 Dialogue Management 193

7.2.2 Automatic Feedback Estimation 194

7.2.3 Complex Action Learning 195

7.2.4 User and System goal achievement 197

7.2.5 Socially Intelligent Systems 198

7.2.6 User Simulators and Evaluation 199

7.3 Conclusion 201

7.4 Impact On Society 202

REFERENCES 203

BIOGRAPHICAL STATEMENT 230

LIST OF FIGURES

Figure Page

1.1 Example architecture of a Dialogue System 2

1.2 Proposed architecture for robotic assistant [1] 48

2.1 Percentage of coverage versus number of agents when using a Gaussian map 56

2.2 The top image shows a random starting state of the system when using a Gaussian map. The middle image shows the state of the system after the execution of the EMSDC algorithm. The bottom image shows the final state of the system after a change in the Gaussian map has taken place and the system has converged to a new solution 57

2.3 Percentage of coverage versus percentage of agents failed, for a static and dynamic system. We can see that the dynamic system adapts well to failures 58

2.4 The various modes of operation of our EMSDC web-based tool 62

2.5 The three phases of the proposed System Monitoring Tool 65

2.6 The proposed architecture for item recommendation. 68

2.7 The proposed architecture for weighted tagging and rating 70

2.8 Sense Graph of a tag. The tag is at the bottom leaf of the graph and the most abstract sense (Entity in most cases) is the root 74

2.9 The Tag Sense Graph for a small database 77

3.1 A Reinforcement Learning agent, interacting with its environment through actuators and sensors. The policy dictates the agent's actions, while taking into account input from the sensors. 80

3.2 Results of our evaluation. In all figures we show the two best performing and two worst performing algorithms. 94

4.1 A Dialogue System sample architecture. 97

4.2 A sample complex action (a) and the procedure of complex action learning (b) 104

4.3 USE will attempt to find a policy that satisfies both hard and soft constraints, as in the case when the soft constraints are satisfied by the policies enclosed by Goal 1, where the algorithm would converge to a policy such as S_1. When no such policy exists, as in the case where the soft constraints are satisfied by Goal 2, the algorithm will find a policy such as S_2 that satisfies the hard constraints and is as close as possible to also satisfying the soft ones 109

4.4 The learning architecture of our Adaptive Dialogue System 112

4.5 (a) Average Reward for various problem dimensions, with confirmation actions ($ASR_n = 0$, $e_c = 0.75$). Optimal is achieved by an already trained system. (b) Percentage of the available rewards achieved for various problem dimensions 120

4.6 Learning curves on a SFP with confirmation actions (7 slots and 4,782,968 complex actions), averaged over 100 runs 121

4.7 Average reward of HSARSA, HSARSA-h and AWL-HSARSA with varying confirmation probabilities (a), varying ASR error level (b) and varying both parameters (c) 122

4.8 Results of USE with one goal (a) and two goals (b) for various changing independent goals 123
4.9 Results of USE with two variables. Figure (a) depicts part of the policy space, where the rectangle is a projection of the optimal policy space onto the goal space, where variables A and B are realised. Goals are marked as g1, g2, g3 and g4 and final system states are marked as s1, s2, s3 and s4, respectively. Goals are represented as crosses and final states as circles. Figure (b) depicts the learning curve and the yielded values of A and B as episodes progress 125
4.10 (a): The grid world, where 'S' represents the starting state, 'F' represents the goal state, 'R' represents states that offer rewards and 'T' represents states with traps. (b): Depiction of paths that evoke certain emotional states 128
4.11 Complex actions that can be generated by AWL for the object manipulation system 133
4.12 Simulation of PR2 manipulating an object 134
5.1 Screenshot of the Museum Guide ADS 139
5.2 Architecture of the proposed Museum Guide System 140
5.3 The architecture of our system 149
5.4 Sample complex actions our system is able to learn. Squares denote physical actions, hexagons denote verbal actions and circles denote complex actions. Rhombuses are only used in this figure in favour of compactness 152
5.5 The architecture of the dialogue manager 153
6.1 Screenshot of the PTSD Assessment ADS 165
6.2 Example complex actions 176

6.3 The architecture of our Adaptive Dialogue System, for assessing PTSD score. Spoken input is captured by the Automatic Speech Recognition module, which transforms it in textual form, which is then intercepted by the Natural Language Understanding and Emotion Recognition modules, which extract useful information. This information is passed on to the Dialogue Manager, which decides how the system should respond, and this response is translated into spoken output by the Natural Language Generation and Text To Speech modules 179

LIST OF TABLES

Table Page

3.1 Online RL algorithms we selected for evaluation in the dialogue policy learning problem. 92
4.1 Update rules of our user simulation model, from [2] 115
4.2 Parameter settings for experiments with varying user simulator behaviour. 117
4.3 Target emotional states and associated dialogue policies, where "L" stands for *moveLeft*, "R" stands for *moveRight*, "F" stands for *moveForward* and "E" stands for *informEnd* 129
4.4 Basic verbal and physical actions, available for our system 132
5.1 Average learning speed 144
5.2 Statistical significance, where $\star$ denotes $p < 0.0001$ 145
5.3 Basic actions available for our ADS, out of which complex actions will be generated. 150
6.1 A list of the most prominent PTSD symptoms [3] 166
6.2 System and user actions, defined in our system 173
6.3 Usability inspection of our system 187

CHAPTER 1

Introduction

While extremely beneficial and important, recent medical and technological advances inevitably lead to an increased population of wounded warriors, injured people, people with mental or physical disabilities and elderly. Many of these groups might require cognitive or physical rehabilitation and all groups require assistive technology to perform their daily activities. While such technology exists and undoubtedly improves their quality of life, interfacing with or tuning it is often cumbersome. Adaptive Dialogue Systems (ADS), which are intelligent systems that employ multiple modalities to communicate with the user, can, therefore, provide an ideal interface between people and technology. Moreover, ADS may be adapted and not use speech as the primary means of communication but use other channels, such as gestures or input from various assistive devices. In the rest of this section we elaborate on ADS, we present the unique challenges faced by an ADS deployed in an ALE and we discuss why we believe ADS can benefit ALE users.

As mentioned before, Dialogue Systems (DS) can be seen as multimodal intelligent systems, able to have a conversation with their users. A typical DS is composed of many modules, such as Automatic Speech Recognition (ASR), Natural Language Understanding (NLU), Dialogue Management (DM), Natural Language Generation (NLG), Text To Speech (TTS) and others, as shown in Figure 1.1. Such systems are becoming increasingly popular in the industry, by acting as service providers, mostly because they are efficient in terms of cost and maintenance. Typical applications include customer support, providing information or movie and flight booking. Adaptive

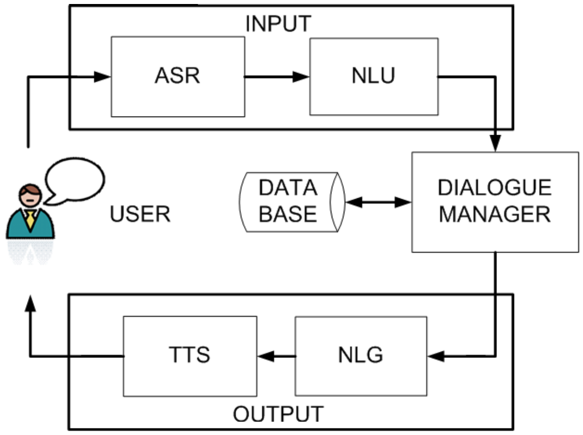


Figure 1.1. Example architecture of a Dialogue System.

DS (ADS) are systems that can adapt to different users, by learning about and remembering their preferences, needs, etc. and can react in real time to environmental changes. Researchers have focused on achieving adaptation in the several modules of an ADS, such as achieving adaptive Referring Expression Generation [4], achieving adaptive NLG [5], adaptive TTS [6] and, last, a large part of the research community has focused on achieving adaptive DM.

Assistive Living Environments (ALE) are environments where the users are, by definition, not competent with technology. Assistive Technologies, therefore, have been developed to aid them in their daily tasks and improve the quality of their lives. While in some cases interfaces for such technologies are designed to be relatively easy for ALE users, tuning them to each user's changing needs may prove really hard. ADS can bridge this gap and provide an intuitive interface that allows easy access to assistive technology. The user would only need to express his / her needs in a natural manner and the system will be able to understand and meet those needs. Moreover, ADS can make technology not designed for ALE available to ALE users, by automatically controlling it as well as handling parameter tuning.

There has been a lot of work in Human Robot Interfaces (HRI) designed for robots operating in ALE, but not much from the ADS perspective. Regarding ADS applied to ALE, the majority of the research community either uses simple and static DS, or just apply speech recognition for vocal commands. While there has been work on the requirements of ADS for ALE [7], in most research efforts ADS or DS are simply another component of the system, rather than its driving force. In the following chapters we propose methods to fill this gap and focus on adaptation techniques specifically tailored to the needs of ALE. Before proceeding, however, to present our approach, we review recent work in both ALE and ADS.

1.1 Recent Advances in Assistive Living Environments

Assistive Technologies is a rapidly rising field, with many branches, such as assistive robotics, intelligent interfaces (such as ADS targeted for ALE), novel rehabilitation and monitoring methods (including purpose-build hardware), and many others. As mentioned before, ADS have the potential to act as mediators between ALE users and novel technology, necessary to improve the quality of their lives. Moreover, ADS can enable those users (which by definition are less competent with technology) to use existing technology, by providing a natural interface (for example spoken or written natural language). This can have a large positive impact on people's lives, as it can greatly reduce the cost of living, as, for example, instead of designing bespoke equipment, an ADS with appropriate interface capabilities can interact with the user's preferred modality (e.g. speech, gestures, etc.) and control existing technology. In the following Sections, we present recent advances regarding ALE users' perception of technology and its role in their lives, how robots are currently used in ALE, sensor networks for ALE and, last, other state of the art approaches and ALE frameworks.

1.1.1 Users' Perception of ALE

Due to the increased interest in ALE, from various disciplines, many studies have been conducted to either determine the needs of various types of ALE users (such as elderly, disabled, injured) or their perspective on having assistive technology in their lives. Sun et al. [8] focus their study on elderly people and try to determine some of the challenges and open problems in ALE. They identify a lack, in current approaches, of fully expressing the power of human users as well as not taking into account the users' social activities and existing connections. Greenhalgh et al. [9] performed a study to address three main questions, namely what elderly ALE users consider important, what effects does illness have on the user's ability to interact with technology and attempt to measure the current presence and penetration of ALE technologies in our society. The authors interviewed 40 elderly people (between the ages of 60 and 98), in order to create a detailed account of their everyday lives as well as determine the extend of technology use. They analysed their data and concluded that most users suffered from co-morbidities, which were gradually worsening and their needs needs were, in general, diverse. More than this, the users expressed negative feelings, regarding their future prospects. Perhaps the most important conclusion of this study is that assistive technology was not able to meet the users' needs, except in very few cases, where, it should be noted, technology was customized to those specific users.

Harrefors et al. [10], on the other hand, examine how healthy elderly people perceive assistive technology, as its use has increased due to the increase in at-home care services. The authors performed their study by conducting interviews on 12 elderly couples, with ages above 70 years. Their results showed that, in general, assistive technologies are regarded as an opportunity, although with unpredictable consequences. The authors emphasize the fact that ALE researchers and professionals need to focus on building trust between assistive technology and its users (especially those with severe medical conditions) and keep human caregivers in the loop.

Alwan et al. [11], also conducted a study, regarding user's perception of monitoring technology as well as its impact on a psychological and social level. The authors installed assistive monitoring devices in 22 homes, to track the daily activities of residents and identify critical conditions. The authors' system was able to produce alerts and activity reports, which were forwarded to caregivers, who also participated in the study. Their input was helpful in assessing how well the collected data could be used by medical health professionals for diagnosis. The results of this study show that residents were willing to accept such technology, which would benefit their daily living. In another study, conducted by Alwan et al. [12] on 15 assistive living homes, the authors again aimed at understanding the psychological and social effects such technology can have on users. The residents' daily activities were monitored constantly and alerts where raised when abnormal behaviour was detected. Reports and alerts were forwarded to professionals, who had agreed to participate in the study and the results were consistent with the previously mentioned study.

Fuchsberger et al. [13], focus on the needs of elderly people and how assistive technology can be used to meet them, while taking into account technological advances (such as media and communications) that can have a great impact on their quality of life. The authors investigate opportunities and challenges of adopting assistive technology, as well as how the actual users (the elderly in this case) should the involved in designing such technology. Agree et al. [14] also conducted a study to measure the effects of assistive technology on the elderly, suffering from chronic diseases. The authors propose an indicator that can capture the effects of assistive technology on the users (specifically the elderly) daily lives. They also found that ALE can have a greater potential benefit for users with greater physical disabilities.

Aloulou et al. [15, 16] conducted similar studies, by deploying assistive technology in a nursing home, using three rooms for a period of 14 months. The authors focused mainly on elderly patients suffering from dementia, while also including professional caregivers in their study. More specifically, 8 elderly patients and 2 caregivers were recruited for this effort, which resulted in a list of requirements identified by the subjects or observed by the authors. The authors' findings include observing increased amount of time being spent by the subjects in the bedroom and washroom, which includes cases where they showered repeatedly throughout the day, the necessity for caregivers to adapt to the different habits and preferences of each subject (in order to provide appropriate encouragement or instructions) and the fact that the subjects' level of dementia affects their suitability to use assistive technology. From a technical perspective, the authors found that before being deployed to ALE, a system should guarantee privacy and security, should facilitate management for multiple people with different roles and should be able to handle failure and adapt to each user's behaviour and preferences.

In order to better understand monitoring data, and especially activity recognition, Virone et al. [17] propose twelve behavioural patterns, derived from the daily activities of 22 ALE users. These patterns can be used in automatic activity recognition monitoring systems, which would raise alerts to professional caregivers upon detecting abnormal behaviour. Last, Kleinberger et al. [18] describe an assistive living laboratory, which can be used by elderly people in order to train and familiarize them with assistive technology and novel interfaces. Such facilities have the potential to be used by both users and professional caregivers to evaluate how fitting and usable specific assistive devices are for specific users or groups of people. In the next Section, we present current advances relating to assistive technology.

1.1.2 Assistive Technology

Due to significant advances in the related fields, as well as their unprecedented commercial availability, robots have been increasingly used in ALE, mostly due to their ability provide assistance through intelligent manipulation of their environment and also because of their potential to engage people in various activities (for example physical or cognitive rehabilitative exercises). One of the greatest challenges a robot faces in a household environment is perception of its surroundings as well as understanding what it is required to do, for example, pour a glass of water, open a door or operate an elevator. To address the challenge of understanding spoken commands from users who are not experts, Fasola and Mataric [19] propose a system that uses Spatial Language Understanding to dynamically infer the spatial relationships between objects in the environment. Lazewatsky et al. [20] propose a Human - Robot Interface (HRI), where an ALE user can guide a robot for table-top manipulation tasks only by movements of his / her head. Fukazawa et al. [21] evaluate and compare multimodal interfaces for ALE, concluding that multimodal interaction using gestures or head movement is equivalent to using a mouse, albeit slower. Autonomous manipulation, however, in a household environment is not yet a commercially viable option, and to address the risks involved, Deegan et al. [22] propose a methodology for designing new technologies for ALE, specifically mobile manipulators. These manipulators would be operated by remotely located professional caregivers, thus greatly reducing the probability of accidents and increasing the users' trust in the system.

Another technology that has penetrated the ALE market, is tele-presence (or mobile remote presence) robots [23, 24, 25, 26, 27, 28, 29]. Such technologies not only allow ALE users to socially interact with their peers, without having to leave their home, but also allow professional caregivers to gain a better perspective of the users' condition. Other assistive devices include eye-trackers [30], designed to aid users mainly in operating personal computers but also other devices (such as wheelchairs [30]).

Perhaps the most important aspect of assistive technology is monitoring and communication through sensor networks. Challenges here include data security, as information stored or transmitted most often refers to monitoring or health data, but also include robustly managing a wide variety of sensors, fusing and storing the available information as well as managing mobile and battery-powered sensors. Virone et al. [31] describe the *Alarm Net*, a context-aware wireless sensor network specifically designed to handle medical health information in ALE. According to [32, 33], *Alarm Net* involves various types of sensors, including psychosocial as well as environmental and facilitates real-time streaming and processing of the data gathered. This network is able to identify daily activities, for example events such as feeding frequency or general activity levels of the users and operates under secure communication protocols to address the heightened need for data privacy. Last, *Alarm Net* performs smart power management to account for battery-powered mobile sensors.

Eklund et al. [34], also propose a wireless network designed for ALE, able to handle heterogeneous sensors. The authors designed their system mainly for elderly ALE users and opted for local data processing, to promote privacy and security. Singh et al. [35], approach this problem from the dynamic trust domain perspective. They investigate the potential of integrating services with an ALE system, while guaranteeing data privacy and attempt to measure trust. Dagtas et al. [36] leverage the extensive use of cellphones as well as inexpensive sensors and well established communication protocols, such as ZigBee or Bluetoth, to facilitate a wireless sensor network for assistive living applications. Their architecture allows for processing data and raising alerts in real-time when critical situations are detected. Last, ALLADIN [37] and INHOME [38] are two other monitoring framework for ALE, utilizing sensor networks. ALLADIN targets elderly users suffering from dementia, and especially the Alzheimer's disease. INHOME focuses more on providing services to ALE users, in a personalised, more efficient manner, and again targets elderly ALE users. In the following section we present other studies and approaches to integration and deployment of assistive living technology.

1.1.3 Other Approaches, Frameworks and Systems

Wigfield et al. [39] investigate the impact of ALE and related technologies in the social care and health sectors workforce. The authors conducted interviews with 32 workers, at various posts, while also collecting relevant reports from each facility. The authors found that the assistive technology used was fairly similar, with differences only in customizing some aspects to each facility's needs. Through this study, Wigfield et al. [39] also identified some challenges, regarding the design of the equipment, its effectiveness and use to its full potential as well as fees imposed by the facilities. Regarding delivery of care within the context of ALE, the authors found that new job roles have arisen, requiring special skills and current roles have been expanded with new tasks and responsibilities. Examples of new job roles include tele-health engineers, installers, evaluators and others, while new skills in existing roles include technical knowledge and better social skills. The authors stress the need for further research in this area as well as the need for a framework dictating the necessary skills that health and social care professionals need to have, in order to seamlessly function within ALE.

Nehmer et al. [40] review some of the challenges posed by ALE, from an engineering point of view and specifically identify the following requirements for assistive technology: robustness, availability, extensibility, safety, security, timeliness, resource efficiency, natural Human - Machine Interaction (HMI) and adaptivity. To meet these requirements, the authors propose that systems designed for ALE should be able to perform self-optimization, self-configuration and self-maintenance. Doukas et al. [41], propose an assistive living framework, able to detect falls through human activity recognition and issue alerts. Their approach collects and analyzes data from audio, video and motion sensors, which are used to track the users of the system. The authors pay attention to the events following a user falling, which indicate the severity of the incident and through advanced methods they are able to determine whether an event should be classified as an emergency or not. An interesting open architecture for addressing some of the engineering challenges of ALE, is I-Living, proposed by Wang et al. [42], according to which an assisted living service provider is responsible for collecting and maintaining data in a secure manner. Such an approach has the advantage of guaranteeing predictable properties and dependable services for ALE integrated systems.

Jara et al. [43] propose an assistive device, targeted for diabetes patients, able to calculate insulin therapy dosage by taking into account many factors, for example illness, physical activity, drug intake and others. Their approach is based on the semantic web to support user profile management through RFID and also to help connect the users with their respective physicians. Kasugai et al. [44] investigate how state of the art display technology can affect the role of HMI, architecture and design of ALE. In particular, they propose to use wall-sized displays to facilitate the illusion of spatial extension and to complement social interactions, such as more realistic representation in video conferences.

1.2 Recent Advances in Adaptive Dialogue Systems

Adaptive Dialogue Systems are DS that are moreover able to adapt to varying circumstances and different environments. DS and ADS have been used for many purposes, such as travel planning, customer support, acting as companions or tutors, and have been increasingly successful for a variety of reasons, such as low associated cost and high efficiency. The research community has been using Machine Learning (ML) methods to achieve adaptation in the various components of an ADS, for many years. Reinforcement Learning (RL) has also been increasingly used, as the RL paradigm fits the interaction very well, and there are many mature and optimal methods we can readily use. Levin et al. [45] were the first who viewed dialogue as an optimisation problem and proposed to model it using a Markov Decision Process (MDP) and apply RL techniques to learn good dialogue policies. Recently, a great part of the research community has been using RL methods to learn dialogue policies online, while interacting with the user, to learn policies for NLG, TTS, and other modules. Before presenting the current trends in ADS research, we first describe how an intelligent ADS should behave, and what are some requirements in order to be perceived as intelligent by its users. The first characteristic of an intelligent system is its ability to fully grasp the meaning of what the user is trying to communicate, not only looking at the utterance, but also at social signals, the interaction so far, shared experience, general knowledge etc. Another important skill is the ability to detect humour, metaphors, irony, sarcasm, etc., as well as knowing when it is appropriate to speak, depending on the system's artificial personality. Of course, an intelligent ADS should understand when it has lack of knowledge and should know how and when to ask for more information. Last, important aspects of intelligent behaviour include common sense, knowledge of important historical and recent events, as well as the ability to lead a conversation and guide it so as to achieve the system's and the user's goals.

In this section, we present notable recent advances in ADS, focusing on state of the art DM and adaptive DM techniques, methods for User Simulation and Evaluation of SDS.

1.2.1 Adaptive Dialogue Systems

An example of the impact of DS in the way people interact with computers are the many spoken dialogue systems serving for customer support. Another example, showcasing how far DS have come, is is Watson [46]. A milestone in Question Answering systems, similar to Deep Blue which won the famous chess player Kasparov, Watson [46] is a DS that won the Jeopardy! challenge.

To make such a system successful, there were many complicated subtasks that needed to be performed, such as parsing, question classification and decomposition, knowledge representation, reasoning and others. The authors also needed to accurately compute Watson's confidence scores, in order to choose the correct answer (among several candidates). To achieve this, each component of their system produces a confidence score, along with its output and these confidence scores are combined by a hierarchical machine learning method to produce the overall confidence score for a specific answer. A score above a pre-defined threshold means Watson will risk answering the question [46]. Other great examples of DS or ADS include Ada and Grace [47], the virtual human twins, which currently reside in the Boston Museum of Science, and can answer questions posed by the public. In the rest of this section, we explore how researchers currently achieve adaptation and personalization in ADS, focusing on one component at a time. We begin by presenting existing ADS architectures or frameworks and proceed with reviewing methods proposed in the literature, for adapting each individual component.

NIMITEK [48] is another example of SDS and specifically is a system designed to help users achieve a task using a graphical interface. An example task is the Tower of Hanoi problem, which the user tries to solve by verbally addressing the system (i.e. without using a mouse/keyboard etc). According to the authors, in order for the interaction to be natural, the system must take into account the user's emotional state. The user must also be convinced that s/he is taking part in the conversation and must not have to think how to phrase sentences in order for the system to understand her/him. A dialogue system targeted to help users, therefore, must not assume that the user will explicitly request help, as users frequently omit information they believe is known by the system. According to [48], there are three important interaction features to better understand users: state of the task, history of interaction and focus-of-attention information. The authors use a focus tree to represent attentional information, where each node (except from the root) represents a sub-focus of its parent. The decision making process addresses three problems: when to provide support, what kind of support and how to provide support to the user.

Cuayahuitl et al. [49] provide an evaluation of a spoken dialogue system, modelled using Semi Markov Decision Process (SMDP) and trained using Reinforcement Learning. They also propose a heuristic dialogue simulator, where the simulated user employs knowledge-rich states and the machine employs knowledge-compact states. A hierarchical sequential decision making approach is followed to dialogue strategy, where dialogue states can be described at various levels of granularity and an action can execute a behaviour using either a single act or a composite sub-dialogue. This is beneficial because a big problem can be decomposed to easier to solve sub-problems, which may only include relevant knowledge and are thus easier (and faster) to solve. Another benefit is that solutions to sub-problems might be reusable in other, larger, problems. The dialogue simulator, proposed by the authors, is able to produce coherent and distorted dialogue acts to be used for training. Their approach assumes action selection is done on dialogue act types rather than dialogue acts, to reduce the space. The authors treat each composite dialogue action as a separate SMDP (an MDP can be decomposed to SMDPs hierarchically). Hierarchical Abstract Machines (HAM) control dialogue behaviour in deterministic state transitions and Reinforcement Learning optimizes decision making. In their evaluation, the authors model ASR errors by distorting dialogue acts with probability 0.2 and randomly assigning confidence levels. They also distort machine acts with probability 0.1 to model user confusion. The results showed that the fully learnt policy could induce infinite loops, but other than that it outperformed semi-learnt and deterministic policies. Fully learnt behaviour was the worst in terms of task success (due to the infinite loops).

Another model for ADS, that extends the Information State model [50], is the Hidden Information State (HIS) model [51], designed primarily for information retrieval tasks. According to the authors, a real-world SDS implemented using MDPs is intractable, as dialogue optimisation requires forward planning, which is almost impossible in a deterministic framework. Moreover, the dialogue manager should have some (complex) error recovery procedures, in order for the system to appear realistic. The authors, therefore, opt for Partially Observable Markov Decision Processes (POMDP), which are able to account for uncertainty during the interaction. POMDPs maintain a distribution across all possible states, rather than estimates of the most likely states, thus allowing multiple competing hypotheses to be considered. This ensures the dialogue manager tracks all possible paths instead of the most likely one. Also, error recovery is simplified since we only need to shift focus to another hypothesis. One solution to this problem would be to use slot-filling approaches, where the complete dialogue state is reduced to the state of a small number of slots that need to be filled. This approach can allow some dependence between slots using dynamic Bayesian Networks. Another interesting approach is to retain a full state representation but only maintain probability estimates over the most likely states [52]. We can see this as having multiple dialogue managers and at each turn selecting the one with the highest likelihood to be representing the true dialogue state. HIS groups similar states into partitions and maintains a single belief state for each partition. To obtain the optimal policy, we can discretise the belief space and then apply standard MDP optimisation methods. Grid points can be concentrated on regions likely to be visited. At any arbitrary state, the closest belief state is selected and its action is used. This could, however, lead to errors and thus the distribution of grid points on the belief space is very important. The main idea of the HIS model is that at any point in the dialogue, we have no evidence for most (of the possible) user goals and therefore these states have identical beliefs. HIS therefore groups these states into equivalence classes, where members of each class are indistinguishable. These classes are called *partitions*. Initially there is one partition and as the dialogue progresses this root partition is repeatedly split into smaller partitions, in a binary fashion $p \rightarrow \{p', p - p'\}$. Multiple splits can occur at each time step and beliefs are computed on the partitions rather than individual states of S_u. Whenever a partition is split, its belief mass is reallocated accordingly (initially the root partition has belief $b^0(p^0) = 1$). HIS also assumes that changes in the user's goal will be explicitly signalled by the policy decision process $P(s'_u|s_u) = \delta(s'_u, s_u)$.

More specifically, HIS represents dialogue states as triples $< p, a_u, s_d >$ that model a hypothesis regarding the true (hidden) state of the dialogue system and maintains a list of M most likely hypotheses to approximate the distribution. A hypothesis in HIS is composed of a specific user goal partition, the last user act and specific dialogue history. It should be noted here, that there may be several distinct dialogue history states, corresponding to different interpretations of the user's input. HIS uses a partially instantiated tree to efficiently represent partitions and, thus, partitions of the user's goal space are represented by a forest. Splitting a partition, therefore, means selecting an uninstantiated leaf node and splitting it so that once copy is unchanged and the other is expanded according to ontology rules. Probability and belief masses are allocated accordingly. In the case of non-terminal nodes, $P(p'|p)$ is specified as a prior in the node expansion rules. The authors stress the fact that this belief refinement is distinct from belief monitoring and that the probability of each hypothesised user act plays no part in the splitting process. The quality of the N-best list of hypotheses (generated by ASR) is crucial to robust performance of the observation model. HIS represents policies by a set of grid points in summary belief space. Beliefs in master space are mapped in summary space and then into a summary action via a dialogue policy. This summary action is then mapped back into master space and output to the user. Machine acts are also mapped to summary space and the necessary items are then inferred (when mapping back) by inspecting the most likely dialogue hypothesis. The dialogue policy can therefore be represented by a set of grid points and an action at each point. We do need to define a distance metric between grid points though [51].

In the following sections we present recent advances in some important ADS categories, such as affective DS, companion DS, robot DS and socially intelligent DS.

1.2.1.1 Affective Dialogue Systems

Affective DS (AfDS) are DS that have the additional ability to model emotion (of the user and the system) and can adapt their behaviour accordingly and in this Section, we present research efforts aimed at creating such systems. An AfADS typically is a multi-modal system, where the user's affective state is monitored via multiple modalities and on the other hand the system employs multiple modalities to express its own affective state (emotion). We cannot really claim that computers will become intelligent without emotions. While this has been backed by many researchers, a more realistic goal of AfDS is to appear more human-like by understanding and simulating emotions and being able to form of long term relationships with their users.

Bui et al. [53], propose a model based on POMDPs (factored POMDPs) for an AfDS. In such systems the main challenge is how to infer the user's affective (emotional) state and adapt to it while taking into account the system's goals. Automatically assessing the user's affective state has many benefits, however, affective state recognition results are typically ambiguous or uncertain so in a real application it is very difficult to assess the user's affective state. Moreover, this state changes over time and the model must take this into account. The authors' proposed system uses POMDP for an affective dialogue model, as POMDPs can incorporate the affective state, intention and other hidden components of the user's state, they can deal with uncertainty and last we can use this framework to ("easily") create simulated users and train and evaluate a system. The authors believe that a model that factorizes states and observations is a good way of describing the dialogue state, which, in the proposed model, consists of a user goal, a user affective state, a user action and a user grounding state. The observed user action an observed user affective state form the observation state set of the model. The authors' transition model consists of a user goal model, a user emotion model and a grounding state model.

Bui et al., [54] employ real-time belief update and online action selection methods to create a tractable AfDS. They follow the Dynamic Decision Network (DDN) - POMDP approach, where each slot is modelled as a POMDP and then approximated by a DDN [55]. The authors split the dialogue manager into two parts, the slot-level, where a slot is modelled as a factored POMDP and the global dialogue manager which is hand-crafted. To represent the emotional state of the user one can either have a discrete variable, for example $E_u = \{stress, frustration, hapiness, ...\}$, or a sub-network of two continuous variables, valence and arousal (according to psychology). The flexibility in the modelling of complex user emotions is one of the advantages of this proposed system. The observed emotional state of the user, E_u, can be modelled by features like speech pitch and volume, body posture, gestures, facial expressions and others. The authors approximate each POMDP (corresponding to a slot) with a kDDN (DDN with k slices) and tractability is achieved by limiting the number of look-ahead steps. The authors also propose a model for user simulations. This model is goal-driven and each user action (although selected based on a distribution) is generally consistent with those goals, except in case the simulated user is stressed and makes a mistake. Apart from robustness to recognition errors the authors stress the need of an AfDS to be robust to user mistakes as well, for example in emergency situations when the users are stressed. They evaluate their system in a tunnel evacuation scenario where the system needs to navigate trapped users to safety. It should be noted here that the authors assume that when a user is under great stress he/she is more likely to make a mistake in response to an *ask* action from the system rather than a *confirm*. Moreover, they have tuned their model so that the simulated user's stress level changes gradually as he/she interacts with the AfDS. The authors tested and evaluated the proposed model's belief update time using a system with 1000 slots and 10 values per slot. The number of slots does not affect the complexity of the kDDNs (which does depend on the number of values though) and this is the reason why the system can cope with many slots. Last, the proposed model cannot perform online belief updates online, since time increases exponentially with the number of look-ahead steps. In many-valued slot problems (slots with thousands of values) the authors propose to use a list of numbers plus *previous* and *next*: $\{one, two, ..., ten, previous, next\}$ and let the mapping between those values and real values (with meaning) happen real time during the dialogue.

1.2.1.2 Companion Dialogue Systems

A very important and upcoming branch of dialogue systems are Companion dialogue systems. According to [56], these are systems acting as collaborative and conversational partners that employ novel HMI models to allow the user a higher degree of initiative. Companion DS should be able to perform a broad range of tasks, rather than being purpose-built, and some of those tasks may be perpetual (such as maintaining a relationship with the user). Webb et al. [56], propose a Wizard of Oz system that generates companionable dialogues. Companionable dialogues are conversations where the system responds to the user's utterance according to his/her emotional state, and follows certain types of strategies such as "positivity" or "empathy". Using such data, collaborative dialogue models can be built. The authors propose this system to generate some examples and later be able to evaluate such conversations resulting from a real dialogue system. The evaluation of a companion dialogue system, however, requires new metrics, as, for example user satisfaction or task completion are not enough (a task may never be completed - "maintain a relationship with the user"). It has been shown that people tend to form relationships with technology, especially if there is some sort of character or avatar assisting them in the use of technology [57]. Relationships take some time to build and are personal, persistent and emotional. The authors use three strategies for their experiments, empathy, positivity and adaptive. Empathy means the system tries to mirror the user's mood (eg ignore any negative input when the user's overall mood is positive), positivity means the system tries to make the user's mood more positive and adaptive is a combination of the two. The system also needs to evaluate the user's mood and, in this respect, it estimates the sentiment of the user's utterance, on a coarse scale and subsequently calculates a score. It then sums up all scores and gets an estimate of the overall mood of the utterance. The system then will choose a technique (independent of the dialogue policy) to respond to the user, such as by temporal order, by recency order, target a particular utterance or prioritise the utterances and address them in that order.

Sieber et al. [58], propose a model for episodic memory, that can be used as a module of a companion dialogue system and is able to remember long term user preferences and interests and comment on past interactions. In order to create long term relationships with users, the system needs to remember past interactions (and other history). The authors propose a system with short-term memory (current user utterance and currently important things) and two types of long-term memory, semantic (elements of the ontology the user is interested in) and episodic (agent experiences, utterances-actions-evaluation of those actions). To retrieve episodes from its memory, however, the system needs a similarity measure. Wilks et al. [59] present a companion DS, designed to interact with elderly users. This system is designed to elicit information from the users, through conversation, regarding photographs depicting events in the user's lives, aiming to compile a life narrative of each user. The authors use ML techniques to detect emotion, from lexical content as well as speech features and employ a DM that allows mixed initiative during the conversation.

Adam et al. [60], present toy companion (or a virtual agent [61]), designed to interact with children and propose a framework to elicit, represent and later use information gathered from the child, during the interaction. The authors achieve personalisation of the conversation by first analyzing dialogues between adults and children to retrieve the strategies for personalisation adults use and subsequently creating personalisation strategies. More specifically, they identified the following strategies, used by the adults: relation to self, personal questions, child control, reciprocity, recalling shared activities, child's preferences and child's agenda. Based on those, the authors split the interaction in two phases, information gathering and information exploitation, and created several strategies for each phase. In the following section, we describe a rising sub-field of DS, which uses robots as embodied agents to engage users in conversation.

1.2.1.3 Robot Dialogue Systems

As robots become safer and more accessible to people, an increasing number of researchers are focusing on developing the necessary technology that will render robots safe and usable enough to be used as household appliances. A very important aspect of this effort is HRI, which largely depends on DS, giving rise to the field of Robot Dialogue Systems (RDS), i.e. DS (or ADS) specifically engineered to interact with users in the physical world. There are numerous challenges in this task, besides safety, such as perceiving the environment and interpreting it as well as coordinating verbal actions with physical ones (e.g. move the arms while talking).

Cuayahuitl and Kruijff-Korbayova [62], take a multi-agent approach to HRI, using hierarchically structured RL agents. The authors use the NAO robot, from Aldebaran [63] to create an embodied agent that will interact with children by playing a game of quizzes. They employ flexible hierarchical dialogue control [49] to handle the interaction and express both verbal and non-verbal behaviour. The same authors, in another work [64], present their approach to dialogue policy learning, which takes into account uncertainty deriving from speech as well as gestures. To achieve this, they distinguish the user's input in to categories: verbal and non-verbal. The system, then, updates its belief state based on both types of user input, and the DM subsequently decides makes a decision based on the updated belief state. This approach, to jointly optimize speech and visual beliefs outperformed dialogue policies that were separately trained.

Kruijff et al. [65], use a factored MDP to try to determine at runtime how much (speech) input needs to be processed. The goal is to decide online which procedure (for analysis) to apply to which part of the input. The controller may select a combination of procedures as well. It decides based on the expected value of running a process and the potential cost it incurs. The authors provide an example of a robot learning colours by example ("this is blue", etc) and making the connection.

Meena et al. [66], propose a chunking parser designed for semantic interpretation of spoken navigation instructions, in English. Their approach outperforms the baseline and is robust to speech recognition errors. Tellex et al. [67], attempt to address some RDS challenges, such as ambiguity in natural language input or limited perception of the environment, by proposing an information-theoretic approach, which enables the robot to ask for clarification, and subsequently exploit information from the user's answer to perform disambiguation. Azhar et al. [68] address a similar problem, specifically of using dialogue to recover from errors due to miscommunication, using an argumentation framework that allows the robot to maintain belief states while providing reasoning functionality using those beliefs as well as conflict-resolution. The latter compares the robot's belief states with the (estimated) human user's belief states to detect conflicts, and attempt to resolve them.

Guiliani et al. [69], describe a robot, able to communicate with human users and collaboratively build wooden toys. The authors use a system to infer the users' goals as well as detect their errors, by exploiting verbal and non-verbal input. The system then uses situated references methods to explain the errors to the users and provide them with possible solutions. Foster et al. [70], evaluate this RDS, with promising results.

1.2.1.4 Socially Intelligent Systems

As the fields of ADS, Robotics and other related research areas progress and, equally importantly, as the cost of hardware decreases while its performance increases, social skills in intelligent systems become more and more important. Robots, as discussed in the previous Section add another dimension to the interaction, creating an even more pressing need for social skills in intelligent systems. Social skills have many aspects, such as emotion recognition or expression, artificial personalities, how a robot should approach a group of people or an individual, how polite should an intelligent agent be, whether to use formal or informal language, and many others. In this section, we present the state of the art in socially intelligent systems, which use ADS to handle the interaction.

Kruijff [71] proposed a model to manage trust between teams composed of robots and humans, taking into account social sentience, i.e. each member's perception of its social role, and the social dynamics of the group. Ochs and Pelachaud [72], conducted a study to assess the impact that two types of a virtual agent's smiles (polite and amusing) have on the user. The authors subsequently propose a model to predict the effect a smile might have on the user. Bevacqua et al. [73], present a model to influence interlocutors through conveying information via audio-visual communication back-channels (such as nodding, saying "ah-uh", etc.). De Sevin et al. [74], propose an online algorithm to select which back-channel signals a virtual agent should express, based on its personality. Prepin and Pelachaud [75] model interlocutors as dynamic agents, that are coupled according to the oscillator model, a trend in dialogue management, where turn taking is an emerging phenomenon rather than happening due to the model itself. The authors assess the effects of time delays on the interaction, either between human-human or human-agent and attempt to quantify the maximum acceptable delays, as well as parameters that influence that maximum. Konstantopoulos and Karkaletsis [76], propose a method for controlling how the system's personality is perceived by its users. They use a personality module to regulate how system actions are realised, based on various parameters, such as the current user model, the estimate of the user's emotional state, the goals of the system, and others.

Bergmann et al. [77] present results of a study, attempting to identify how people use gestures to regulate dialogue. Such a research has the potential to enable social robots or avatars, controlled by DS, to be more active in the conversation and convey meaningful messages through gestures, for example acknowledgement of the user's input or indication that the system does not understand. Another interesting study on gestural alignment in natural dialogue, meaning the interlocutor mimicking a gesture as a means of acknowledging, was performed by Bergmann and Kopp [78]. The authors were also aiming to assess whether gestures are influenced by interlocutors, as well as how they are aligned with speech and communication in general. Last, a related field that has recently emerged, is social signal processing, whose purpose is to model and interpret communication signals, as well as define ways to express them, aiming to create socially intelligent agents [79].

1.2.2 Dialogue Management

In this Section, we present what we have identified as the current trends in ADS research, from the perspective of DM. In the following Sections we elaborate on state of the art methods for dialogue policy learning, modelling complex system actions, automatically estimating environmental feedback, exhibiting social intelligence and modelling simulated users, applied for training and evaluation of ADS.

According to Bohus et al. [80], the most widely used models for DM are finite state machines, slot-filling, information state update and plan based and propose RavenClaw DM, which belongs to the last class of dialogue managers. It is a hierarchical plan based DM and separates domain dependent from domain independent parts of the dialogue management procedure. Concerning dialogue policy learning methods, there is a rapidly increasing number of DS and ADS that employ POMDP to account for uncertainty and apply (belief) state tracking methods to estimate the user's goals or the current dialogue state. Such systems are frequently referred to as Statistical Spoken Dialogue Systems (SSDS), and usually apply online RL methods to learn optimal dialogue policies, while the system interacts with users, and function approximation techniques to account for the state-action space scalability problem, pertinent in RL methods. Examples include [51, 81, 82, 83, 84, 5, 85, 86, 87, 88] and we will elaborate on these efforts in the following Sections. Models that usually do not use Markovian structures to represent the interaction, include the Information State Update (ISU). A mixed-initiative DM, which employs a forward looking method for inference and adheres to the ISU paradigm was proposed by Morbini et al. [89].

Thomson et al. [90] propose a novel framework based on loopy belief propagation, called Bayesian Update of the Dialogue State (BUDS) for updating the dialogue state and learning dialogue policies. The authors extend the Loopy Belief Propagation (or Sum - Product) algorithm [91] by grouping concept values and assuming little changes in the user goal. They also do policy learning with the episodic Natural Actor Critic (NAC) [92] algorithm, over a factorised state space. While the user goals can also be factored into sub-goals, dependencies on these sub-goals must be limited to ensure tractability. The authors build on the HIS [51] idea of grouping states into partitions and apply it to Bayesian Networks and factor graphs. They use a two-time-slice approximation to make the system tractable, meaning they do not keep history that is more than two time slices old, which has been proven to be a good approximation. While HIS gives the exact update for the belief state, this approach yields an approximated update. The authors also use *Summary Actions*, to reduce the complexity and the system then maps these actions to normal actions. They use a softmax function for policy learning and factor policy π parameters by using a basis function for each concept. This system was evaluated through simulations and user trials and outperformed MDP and Finite State systems. However, the effects of policy learning using BUDS were inconclusive.

According to Crook et al. [87], there are several restrictions imposed by POMDP independence assumptions, more specifically current POMDP Statistical SDS (SSDS) with state approximation make it impossible to have sets of goals, which may possibly negating one another. Goal sets, for example, may represent trade-offs that the users are interested in. Moreover, POMDP systems treat any spread of probability mass as uncertainty that needs to be resolved and thus force the user into one specific goal (eg one type of food only). To address the first challenge, the authors propose a method, named Automatic Belief Compression (ABC) and relax those independence assumptions and introduce user goal sets. The users' goals are not directly observable, so there is need to model uncertainty in the model and this is one of the reasons why researchers use POMDPs. The authors assume that there is a single fully constrained user goal and that the only uncertainty is that of speech recognition errors. Another challenge is that POMDP state spaces can grow very quickly, as POMDP belief state is an N-dimensional real valued variable, where N is the number of dialogue states (i.e. the belief state is a probability distribution). Current techniques use hand-crafted rules to compress the state space, assuming several independencies (e.g. the user is interested in one type of food only). The authors suggest to use ABC to reduce belief state dimensionality, while preserving useful distributions within the space.

Another trend in this area is incremental dialogue, where the smallest *conversation unit* is defined as the smallest part of an utterance that can be translated into a dialogue act. This, therefore, defines *micro-turns*, allowing many of the aforementioned problems to be viewed from a completely different perspective. Among other benefits, the DM can begin processing the user's input, while the user is still speaking and take action if necessary (interrupt, provide information through back-channels, etc.). A combination of various modalities can inform the users of how their intentions are perceived by the system, giving them a chance to correct the system early, and allow for more natural interaction [93]. Dethlefs et al. [94], apply RL to learn how to combine multiple output modalities and investigate how verbal and non-verbal actions can be combined to achieve more natural interaction and fewer system self-corrections. Incremental speech and natural language understanding is also trending [95] and will be discussed later in this Section.

One great challenge of RL methods is designing a good model of the environment, able to provide meaningful feedback to the learning algorithm, as the performance of RL methods, including those described in the following sections, heavily depend on this aspect. There have been efforts to tackle this problem, mainly by applying Inverse RL, whose objective is to learn a good function than models environmental feedback. Asri et al. [96] propose a methodology that uses reward shaping to learn a reward function, from dialogue data.

Aiming to provide flexible and adaptive DM, we have developed an adaptive DM framework, which supports the most prominent DM found in the literature, namely Finite State Automata, Slot Filling / Information Seeking, Plan Based and Information State Update. We use a universal structure, Partially Observable Semi Markov Decision Process, able to transparently transform to any other Markovian process, allowing the designer the flexibility and support to implement DM techniques proposed in recent literature. One of the benefits of our approach is that it allows modular DM, where designers and non-experts in ADS development can select components that fit their needs and easily create effective DM. We present this framework in detail in Section 5.4. In the following Section, we will present recent advances in dialogue policy learning, complex action learning, estimating the dialogue model's parameters and representing goals.

1.2.2.1 Dialogue Policy Learning

In this Section, we review in detail recent work in dialogue policy learning, since this thesis is largely focused on novel online dialogue policy learning methods. The current trend is to apply Reinforcement Learning (RL) methods to learn good (or even optimal) dialogue policies. The first work that proposed a framework to model dialogue as a Markov Decision Process (MDP) was that of Levin et al. [45]. The authors propose a stochastic model (MDP) for human computer interaction, that formally describes a dialogue system in terms of its state space, action set and strategy. The system's state describes all knowledge the system has about its internal and external resources. The authors propose to state the problem of dialogue strategy design as an optimization problem. The objective function could be a linear combination of several terms representing the cost of particular dialogue dimensions. The state space (in a slot filling dialogue system) is composed N binary variables (one for each slot) that are 1 if the slot is filled or 0 if not. The action set, similarly, has N variables where each variable represents a question about a specific slot. This model allows for quantifying policies. The paper includes two example dialogue systems (one slot filling and one mixed-initiative database retrieval) modelled as MDPs. According to [90], recent research in dialogue policy learning follows two main avenues: one that models dialogue as a conversational game and the system tries to find the optimal action; and one that tries to model uncertainty in the dialogue. Bayesian Networks are often used in the second approach and researchers have also tried to combine the both research directions by using POMDPs.

Following this model, researchers have been applying RL to automatically learn optimal dialogue policies, either from data or online. RL rewards model environmental feedback (i.e. measure the policy's success) and are most often associated with dialogue length and task completion. There are two main learning approaches, namely the model-based approach and the simulation-based approach. The first estimates the state transition probabilities from a corpus of dialogues and then via Dynamic Programming computes an analytical solution. The second approach uses a stochastic user simulation, trained on dialogue corpus data and a RL algorithm that learns from sample rewards. User simulations represent utterances in an abstract manner, as dialogue acts, which makes it easier to simulate errors [97].

Lemon et al. [84] were some of the first to propose a system (or framework / development platform) for developing and optimizing spoken adaptive dialogue systems that apply hierarchical reinforcement learning. Cuayahuitl et al. [98], later use SMDP to model the dialogue problem and a hierarchical reinforcement learning approach to find optimal dialogue policies in a spatially aware SDS. Their approach is to learn low level policies off line and high level policies on line. They point out three main challenges in the way-finding domain where their system operates. First, decide which route to present, second, decide level of detail in which the route will be presented and third, decide how to present the results, in textual form, using schematics or something in between. The authors argue that due to the very large space, offline learning is infeasible and so they apply on line learning to find the best route. They model the problem as an MDP and then decompose it to several SMDPs in a hierarchical structure. They then apply a recursive learning algorithm, named *HSMQ-Learning with policy reuse*, that is able to solve hierarchical SMDPs in an online fashion. The authors conclude that the system learns better policies if dialogue and spatial behaviours are simultaneously trained than in the two modules are trained individually. Note that hierarchical learning is an alternative scaling technique to function approximation.

Lemon [5], attempts to jointly optimise Dialogue Management and Natural Language Generation in a single statistical model, using RL. In DS, rewards typically are not immediate but delayed, upon successful (or not) completion of a task. The author applies function approximation to reduce state space size, learning realistic user simulations from small data sets, regression techniques for data-driven discovery of reward functions and hierarchical RL, allowing hand-coded and learnt decisions. Function approximation allows a learned policy to generalise to similar unseen states. His results show a statistically significant 27% increase over a baseline policy, derived without joint optimisation. According to Lemon [5], however, RL with a hand-coded reward function is not much different than hand coded rules. A solution to this problem is regression analysis to discover a linear combination of objective measurable features that are predictive of subjective user ratings. Hierarchical MDPs allow a mixture of hand coded and learnt decisions, reduce the state space size and can also be reusable. The author, thus, proposes to model optimisation and planning of sequences of actions in context with RL techniques. The author provides an example (restaurant) of NLG and dialogue policy optimising, where the top levels of the hierarchy represent DM choices and the bottom levels represent NLG choices. This model allows conflicting DM goals, such as having small dialogues but reliable search results. The author used real users and trained the reward function using a PARADISE-like [99] scheme. For policy exploration he uses a bigram stochastic user simulation and used a hierarchical SARSA with linear function approximation to learn optimal dialogue policies.

Hof er al. [100], propose a strategy to optimize driver's options by adaptation that takes preferences and knowledge into account. The authors propose two "learning" and four "forgetting" models which are then fused and a result in a new model that describes the driver's knowledge and is then used to control the in-car system parameters. To evaluate their approach, the authors asked subjects to drive for an hour in moderate traffic and accomplish 10 tasks. They had to learn the voice commands, and no help was given apart from the 'Options' command that lists available options in each menu. These options are structured in layers, depending on usage, and the system is able to learn commonly used commands. The authors measured the time to complete each task and used power law to describe learning in their environment. It should be noted here, that the authors identify two user categories: Novice and Expert.

Rieser et al. [85], train a RL policy aiming to address the problem of how many database results to present to the user and when to present them. The authors also learn optimal solutions for confirming noisy input, such as voice from a mobile device. The proposed system presents at most 100 items to the user and if it retrieves more it keeps asking for additional constraints. The authors assume that the requested item exists in the database, i.e. do not apply constraint relaxation when zero results are retrieved. To narrow down results, they apply monotonic retrieval, meaning that additional constraints are perceived as "AND" or random retrieval, meaning that additional constraints can be perceived either as "AND" or as "OR". To evaluate their approach, the authors employ simulated users for and an objective function which takes into account task completion, dialogue length and number of presented items. Their results show that RL-based policies are robust under various conditions.

Henderson et al. [101], propose a hybrid model for dialogue policy learning that combines reinforcement (SARSA(λ)) and supervised learning (SL), where the latter is used to restrict the learnt policy to parts of the dialogue space for which we have data. According to the authors, adding contextual features (also including domain information) leads to better dialogue policies, and therefore they use linear function approximation to achieve generalization to states that were not present in the training data. They also note that policy exploration with simulated users is intractable, cannot replace human users and does not solve the sparsity problem, due to the huge policy space and thus, train their system using fixed data sets without policy exploration. The data they use come from the COMMUNICATOR project, where users booked flights and sometimes a hotel and/or a car. The authors define the reward function through user questionnaires and task completion metrics. The approximation method they use treats dialogue states that share features as similar (this measure for similarity is known as *linear kernel*). One of the challenges Henderson et al. [101] faced, due to the fixed dataset they used, is that there is not enough information about some states and this leads RL algorithms to unreliable estimates. More than this, policy exploration in large state spaces (in this case $74^{10^{386}}$) is a huge problem, where RL algorithms cannot scale easily and the vast majority of policies are very bad (i.e. only a small subset of the huge policy space contains optimal or even acceptable dialogue policies). One way to deal with this problem is to begin with a policy known to perform well, and try to adapt or update it. The authors try to tackle the problem of insufficient data through SL, to determine which actions to take when the system is in a portion of the state space for which there is not much information. Their evaluation showed that the proposed hybrid model outperformed purely supervised or purely RL models.

Henderson et al. [52], propose a Mixture Model of POMDPs to represent uncertainty in DS. Each MDP state corresponds to a distribution over POMDP states (belief states). Their system, evaluated by human users, outperformed MDP or handcrafted systems. Lison et al. [102], use a POMDP to represent the interaction and dynamically constrain the action space, based on prior knowledge over locally relevant dialogue structures. The authors opted for POMDP, to address the challenge (mostly for open ended spoken dialogue systems) of the very large and sparsely populated state space, necessary to model the dynamics of the environment. POMDPs provide an explicit account for a wide range of uncertainties, related to partial observability and stochastic action effects. The authors propose a hybrid approach that simultaneously addresses uncertainty and complexity in dialogue management. More specifically, they propose a pruning mechanism that restricts the action space at a given time, aiming to make the planning problem tractable. Their approach has two steps. In the first step they structure the state space using a Markov Logic Network and then use prior knowledge to derive admissible and inadmissible actions, which are then pruned. The second step is to choose the optimal action from the reduced action set. Prior knowledge, in that work, is described by a set of rules defining admissible dialogue moves in a particular situation. Weights of these Markov Logic rules (deciding whether an action is relevant, given the current state and previous action) can be either hand coded or learned from data. The relevant action space is created by estimating each action's probability and selecting those above a given threshold. The authors, extract for each belief state a set of most likely states and compute relevant actions for those only.

Gasic et al. [81], attempt to tackle the problem of tractability in POMDP policy learning and propose to apply Gaussian Processes (GP) and RL. This approach, named *Gaussian Process SARSA*, not only makes learning (in an online fashion) faster but also provides an estimate of the uncertainty. They estimate the optimal *hyper-parameters* (i.e. the GP kernel parameters) by maximizing the marginal likelihood and show that this approach greatly outperforms a grid based Monte Carlo control algorithm in terms of convergence speed, but they both converge to the same performance levels in terms of dialogue policy quality. They also provide insight on how kernel parameters can be trained using dialogue data. Gasic et al. [103, 104] evaluated dialogue policies learnt by this method, when using the BUDS [90] DM and trained on a few hundred dialogues, using human subjects and achieved promising results. Gasic et atl. [105] use *Gaussian Process SARSA* and POMDPs to address the challenge of porting a SDS into a new domain. In such a case, the system typically needs to be re-trained, from scratch. The authors, however, propose to adapt existing (good) dialogue policies to handle concepts of the new domain. Their approach in this proof-of-concept work is to extend the domain by one concept, and adapt their dialogue policy using only a few training episodes. The authors achieve good results and effectively take the first step in addressing this challenge.

Rieser et al. [85, 106], explore several machine learning methods for dialogue policy learning using multi-modal wizard of oz data. They find a uniform multimodal policy across wizards, and their model outperforms the majority baseline. The authors point out that in such multi-modal human computer interaction interfaces of interest is the human wizards' behaviour as well and not only the users'. According to Rieser et al. [85], such data have the potential to help in understanding how humans react when facing certain situations (requests by users etc) and create a model of human behaviour in multi-modal environments. The authors conduct experiments and investigate when human wizards ask for clarifications using speech or other modalities. They then use machine learning techniques to evaluate these strategies and conclude that they are suboptimal. According to the authors, in such multi-modal environments, one should take into account context and user preferences. It should be noted here, that the authors are only interested whether the human wizard used another modality (instead of speech) but not in the specific parameters (table contents, rows etc). During these experiments it is not always clear what caused a clarification request by the human wizards, so the authors asked the wizards to indicate what the problem was. The authors conclude by stating that Reinforcement Learning trained policies significantly outperform policies that mimick human wizard behaviour.

Pietquin et al. [82], claim that online off-policy learning algorithms are better for dialogue policy learning since they alleviate the need for user simulations (used to train online on-policy systems). They apply a sample efficient online and off-policy learning algorithm called Kalman Temporal Differences (KTD) [107], where they have modelled function approximation (used to solve large dialogue problems) as a filtering problem and then applied Kalman filtering techniques. Their results show that KTD can achieve similar performance to Q-learning, FVI or LSPI using very few samples, much fewer than those the other algorithms require. The authors claim that as the user population of a dialogue system evolves, so does the optimal policy π^* and consequently the optimal parameters θ^*. The authors also add centered white noise to their estimation of Q^* to reduce (inductive) bias. They test their algorithm on a slot filling dialogue system, with three slots (too few!!) location, cousine and price and 13 available actions. Off policy learning requires a handcrafted initial policy, but it does not need to be a good one. Not much work has been done focusing on sample-efficient algorithms (algorithms that need small number of dialogues to converge). The authors also note that their results are more reproducible since they exhibit smaller variance. In a different approach, Kimura et al. [108], propose a Weighted Finite-State Transducer based Dialogue Manager (WFSTDM), typically used in speech processing or natural language processing, that is able to handle multiple ASR hypotheses.

1.2.2.2 Complex System Actions

Complex (or composite / hybrid) actions refer to combinations of basic system actions. These may be categorised into *Verbal*, which refer to the system speaking, *Computational*, which refer, for example, to querying a database, and *Physical*, which could refer to controlling the motors of a robot. Cuayahuitl et al. [109] employ SMDPs to represent complex (composite) actions and hierarchical RL to learn dialogue policies. Other approaches include the work of Pietquin et al. [82], where the authors use an online batch RL algorithm to select basic or complex (hybrid) actions, from a pre-defined set of available actions. Complex Action Learning (CAL) refers to learning how to combine basic system actions into complicated ones, aiming to solve more difficult tasks.

An important aspect of executing a complex action is coordination and parallel execution of *Physical* and *Verbal* actions, in a more general sense than lip-speech synchronisation in avatars or robotic heads. Anh and Pelachaud [110, 111], use a repertoire of non-verbal behaviours the agent may execute and, combined with verbal output, the authors aim to achieve expressive story-telling behaviour with Aldebaran's NAO robot. Cuayahuitl et al. [62], use multi agent RL to achieve a similar task, using Multi-Agent SMDPs (MSMDPs) to model *Physical* (non-verbal) and *Verbal* actions, which may be single- or multi-step, and a hierarchy of MSMDPs to model the interaction.

Chandramohan et al. [112], apply the Fitted Q Iteration algorithm to approximate the continuous state space of SDS. Their approach significantly outperforms hand-crafted dialogue policies and a similar system using the Least Square Policy Iteration [113] algorithm for training. Chandramohan et al. [114] identify the the unavailability of dialogue corpora ad one of the major challenges in hybrid action (i.e. complex action) policy learning. The authors advise using simple actions in the presence of noise in ASR as, for example, if a user responds negatively in a 2-slot confirmation action, there is no way of inferring which slot the user is referring to. The authors also note that hybrid actions are more suitable for experienced users and, according to their evaluation, they perform better than simple ones.

1.2.2.3 Model Parameters Estimation for Dialogue Systems

A new way of viewing the dialogue problem is as a parameter estimation problem. In such a setting, one does not care about the specific effects of each parameter, as long as they are tuned to yield the best possible outcome (which is to yield the maximum expected reward in the case of RL). In this Section, we will present an algorithm designed to do just that, to showcase this different approach to training DS.

In the Natural Actor-Critic (NAC) algorithm [92], updates are performed according to the natural policy gradient, which is the steepest ascend according to the Fisher information metric $\tilde{\nabla}_\theta J(\theta) = F^{-1}(\theta)\nabla_\theta J(\theta)$, where $J(\theta)$ is the expected return with respect to some parameters θ and F is the Fisher information matrix. The authors prove that $\nabla_\theta J(\theta) = F(\theta)w$, so $\tilde{\nabla}_\theta J(\theta) = w$ which is the only thing we need to estimate (and not the Fisher information matrix). The policy improvement step therefore is $\theta_{i+1} = \theta_i + \alpha w$ where α is the learning rate.

Jurcicek et al. [115] take advantage of the great properties of NAC, and propose a novel algorithm, called Natural Actor - Belief Critic (NABC), which, based on observed rewards, estimates the natural gradient of the expected cumulative reward and then performs gradient ascend. Dialogue model parameters (i.e. the prior distribution of the dialogue model parameters) and policy parameters are then updated accordingly. Moreover, NABC does not require the user goal to remain constant or the model to be generative. Information about interaction history in the BUDS system is represented as a sequence of actions and observations, which are implemented as dialogue acts. The proposed system deals with uncertainty by providing alternative dialogue acts, each associated with a confidence score. It represents the underlying POMDP using a Bayesian Network, where nodes are slots in the dialogue system. This model can be used for real applications, provided that each slot has few dependencies (and that we use 2-best hypotheses for possible dialogue acts). The BUDS system [90] uses *Gibbs policies* to reduce complexity:

$$\pi(a_t|b(\cdot|h_t;\tau);\theta) \approx \frac{e^{\theta^T \Phi_{at}(b(\cdot|h_t;\tau))}}{\sum_a e^{\theta^T \Phi_{at}(b(\cdot|h_t;\tau))}} \tag{1.1}$$

The proposed system uses summary actions to further reduce complexity. The summary actions available are: ”request”, ”confirm” most likely value, ”select” between the two most likely values. There are also some global actions: ”inform”, ”repeat”, ”bye”. The authors tackle the problem that arises when using policy gradient methods for estimating the parameters of the dialogue model with a hand-crafted function Φ. This function is typically not differentiable with respect to the model parameters τ. The authors then assume that τ come from a prior distribution $p(\tau;\alpha)$ that is differentiable with respect to α and try to learn the parameters α. These parameters will provide an estimate of the prior distribution and then the expected values for τ given the distribution will yield the new estimates for τ. In short, they incorporate dialogue parameters α into the objective function J and estimate the natural gradient $\nabla J(\alpha, \theta) \approx [w_\theta, w_a]^T$. This way policy and dialogue model parameters are trained jointly. They then use w_a and w_θ to update the parameters accordingly: $\theta' \leftarrow \theta + \beta_\theta w_\theta$ and $\alpha' \leftarrow \alpha + \beta_\alpha w_\alpha$. If, for some reason, one wants to train the parameters separately, Jurcicek et al. [83] propose the Natural Belief Critic (NBC), that assumes a fixed policy. NBC, a generalization of NAC, tackles the problem of estimating the parameters of the POMDP model underlying a Statistical Spoken Dialogue System. The authors believe that in most current spoken DS, modelled as POMDPs, the parameters of the dialogue model are set by the designer. Another approach to this challenge, is to apply Expectation Maximisation or Expectation Propagation algorithms, but unlike NBC, they typically require the user goal to remain fixed and the extend to which maximum likelihood correlates with expected reward in a dialogue system. The authors point out that NAC is the most robust algorithm that has been tested with BUDS [90], a fact that implies that natural gradient w of the expected reward is critical. Their evaluation shows that the proposed algorithm converges reliably, but the maximum reward that can be achieved depends on the initialization. Thus, this algorithm works best if one wants to improve an existing set of parameters.

1.2.3 User Simulation

An important part of ADS training and evaluation is creating good user simulators. These can be specifically designed for certain ADS components, for example DM, Referring Expression Generation or NLG, to evaluate how good the system's reasoning and output is. User simulators make training much easier, faster and accurate (by improving generality). There are a number of disadvantages as well though, such as difficulty in assessing the quality of a user simulator or in achieving specific behaviours. Most simulators model the interaction at the signal level or at the intention level. Intentions usually refer to dialogue acts while signals refer to output generated by the simulator, that the ADS needs to process before deriving the intentions.

Moller et al. [116], propose a user simulation model that attempts to simulate user errors. These can be either system driven, conceptual or just user slips. The authors, therefore, classify possible errors into five categories: no input errors, vocabulary and grammatical errors, dialogue state errors (unacceptable input from the user, i.e. a valid input but for another state), system capability errors (unsupported request) and finally model errors (valid input for the current "problem" but wrong system model). The authors show that most errors fall in the second category (vocabulary or grammatical) but $20\% - 40\%$ fall in the rest except no input errors. This reinforces the need to take errors into account, when modelling user simulations.

Engelbrecht et al. [117], propose an automated DS evaluation framework based on probabilistic simulated user behaviour, which is influenced by interface and user features. Rossignol et al [118], apply Bayesian Networks in an effort to detect grounding problems, which occur when the system requests confirmation of a slot value (in a slot-filling DS), while the user has not provided any value for that slot. Chandramohan et al. [119], create a user simulation model using Inverse RL. The authors train their simulator on data collected from human-human dialogues, and learn a reward function that can be subsequently used for dialogue policy learning. The authors of [120] model user behaviour as MDP policies and apply a clustering method to form groups of users that exhibit specific behaviours, aiming to train dialogue managers in a more varied corpus. Chandramohan et al. [121], propose a framework where both the dialogue manager and the user simulator adapt their behaviours during the interaction.

Keizer et al. [122] propose a user simulation model that is based on an agenda and can be trained using real dialogue data. It is hand-crafted for the most part, but can be trained to estimate the optimal parameter values for the model. The authors use a set of decision points to simulate user behaviour and these can either be deterministic or probabilistic. These points then form a network of decision points with weights representing probabilities on the probabilistic nodes. The authors use EM to train those weights, using data from real interactions. Schatzmann and Young [123], present an agenda based user simulation model, in an attempt to address some challenges of this field, such as to create a model that combines conflicting goals. Their approach is robust to noise and errors because it allows simulated users a certain degree of initiative. Goals in the agenda are ordered with respect to priority, therefore this method supports changes in user goals.

Chandramohan et al. [119], approach user simulator training as a learning by imitation problem, and create a user simulation model using Inverse RL (IRL). The authors model the user as an MDP, and, through IRL, train this model to yield similar behaviour as real users exhibit, in a dialogue corpus. Their results seem promising, although this approach was not tested on large data sets. In another work, Chandramohan et al. [120] also model user behaviour as MDP policies and apply a clustering method to form groups of users that exhibit specific behaviours. This approach helps at training dialogue managers in a more varied corpus.

Schatzmann et al. [124, 123] present an agenda-based user simulation model, in an attempt to address some challenges of this field, such as the fact that it is non trivial to create a user simulation model that combines conflicting goals. The authors use a stack or a tree-like representation of user goals, called the user's agenda. A simulated user response is generated by removing items from the agenda (or popping from the stack). This approach is more robust to noise and errors because it allows simulated users a certain degree of initiative. Goals in the agenda are ordered with respect to priority, therefore this model supports changes in user goals. If a new goal arrives with high priority, it will be put higher on the stack (or in the appropriate position in the tree-like representation). The authors also use two summary spaces, where they map dialogue acts and user actions, respectively, in an effort to downsize the vast amount of possible combinations.

Schatzmann et al. [125], provide a systematic approach to user simulation model evaluation. The authors set the goals of a user simulator, which must be able to produce human like dialogue and reproduce the variety of behaviours that a real user exhibits. They propose to evaluate user models by measuring how much the simulated response resembles a real user response and, to quantify this, they use Precision and Recall. The authors also define a set of measures that can be used for dialogue evaluation (and consequently user simulation model evaluation). These include high level feature measures, such as length and amount of information, dialogue style measures such as frequency of different acts and analysis of how many are goal directed or not, degree of user cooperativeness and measures such as success rate and efficiency which help us understand how well misunderstandings are modelled. The authors point out some of the challenges in evaluating dialogue managers when comparing the generated dialogues to real corpora. In the following Section, we review recent advances in evaluating DS, as a whole.

1.2.4 Evaluation

In order to assess the performance of user simulators or ADS in general, a robust and realistic evaluation methodology is needed. Researchers have therefore been trying to measure dialogue systems' performance and possibly develop a universal and objective evaluation framework. The challenge here is that the ultimate measure is user satisfaction, which is subjective and therefore not easily quantified. According to [80], comparing two dialogue systems is like comparing two programming languages. Each has advantages and drawbacks on specific tasks but we cannot exactly claim that one is better than the other in a specific task since the task can be remodelled in a way that both systems perform equally. Before reviewing recent advances in DS evaluation, we describe one of the first generic SDS evaluation frameworks to be proposed, namely PARADISE [99].

The main idea behind PARADISE is that it decouples goals from how they are achieved by the agents. The authors propose the use of Attribute-Value Matrices (AVM) instead of keys, as a way of modelling the expected outcome of a dialogue. These matrices contain attributes such as "Departure City" and all possible values for each attribute, such as "Milan, New York, ...". For task evaluation, the authors use the Kappa coefficient and regarding *dialogue cost*, they represent each cost measure as a function c_i that can be applied to any sub-dialogue. PARADISE tags the dialogue with task attributes and thus links information goals of the task to any dialogue behaviour (making use of AVM), which makes it possible to evaluate different dialogue strategies. It should be noted here, that this tagging may also be used to create a hierarchical discourse structure. All scores are normalized to their standard z-scores to account for different scales (seconds, turns, k etc) and to estimate the Performance function one needs to explicitly ask users for their level of satisfaction and use linear regression to estimate the weight on k and the weights of each cost function. We can also eliminate insignificant cost functions and apply linear regression again. After evaluating two agents, t-tests need to be run, for statistical significance (this can also be done at sub-dialogue level). While there are several other, well known evaluation frameworks for DS, such as SERVQUAL [126], no framework has been unquestionably accepted by the community as able to reliably provide a quantified measure of user satisfaction, across systems and applications.

Moller et al. [116, 127], propose an evaluation framework for SDS that consists of three parts. The first one is a behaviour model that tries to compute interaction parameters, such as system or user errors, confusion, silence and others. The second part is composed of a perception and a judgement model. Both models try to capture the user's perception of the quality of interaction with the system. The last part is the value model, which models environmental parameters such as the system's goal, the environment in which the system operates and others.

Mizukami et al. [128], have developed a prototype SDS on a smart phone, to provide tourist information in Kyoto. The main system is on a server, implementing the Weighted Finite State Transducer Dialog Manager (Hori et al, 2009) while the client application only is on the smart phone. The authors conducted experiments with real users (80) in an office environment, where the users had to interact with the system in a practical scenario (retrieve information about something) and in a free scenario in order to colledt more data on natural language and expressions. After the interaction, each user completed a questionnaire to rate their satisfaction. The authors also measure word error rates, ratio of correct system responses, average system and user delays and utterance length and average user turns per task. They also analysed the subjective evaluations and computed maximum likelihood scores and grouped some questions (from the questionnaire), thus forming the following categories: Acceptability, System Transparency, System Potential and User Comfort. Acceptability was found to have high (positive) correlation with correct responses, meaning users that got correct answers rated those questions highly. The authors also present some of the practical difficulties when dealing with real users.

Pietquin and Hastie [129], provide a comprehensive review of user simulation evaluation for dialogue systems. The authors describe a good metric for simulation evaluation and use its features to assess other metrics proposed in the literature. In short, those features are:

1. *Consistency*: how well the simulation fits the original data statistics
2. *Quality of learnt strategy*: good RL policies
3. *Performance prediction*: how well can it predict an SDS's performance during real user interaction
4. *Generalisation*: ability to generate new (unseen) dialogues
5. *Scalar value*: ability to produce a scalar value to compare with other evaluation metrics or be used as an optimization criterion
6. *Consistent sequences*: assess the simulation's ability to produce (reproduce) dialogue structures and their statistical distribution
7. *Independence*: of task, domain and SDS (or dialogue management system)
8. *Automation*: computed automatically

The authors categorise evaluation methods in two classes, *local methods* and *global methods*. The first ones calculate statistics in turn level and the latter calculate statistics in dialogue level. Local methods, according to the authors, are either distributions or collections of scalar measures and all suffer from *inconsistency* according to the desired features. To summarize, local methods identified by the authors are:

- Dialogue act statistics: relative frequency of acts, user/ system act ratio, ratio of goal-directed vd grounding vs misunderstanding acts, slot values provided upon request (cooperativeness).
- Precision, Recall, Accuracy (event that occurred was predicted with highest probability), Expected Accuracy(average of probabilities the event was predicted with): These methods do not predict *generalisation* well.

- Kullback-Leibler divergence and dissimilarity: $DS(p||q) = \frac{D_{KL}(p||q)+D_{KL}(q||p)}{2}$, this metric is not bounded so it cannot be used easily as an evaluation metric. Also it gives more weight to the means of the distributions rather than the variances and it does not predict *generalisation* well either.

The authors suggest that the above methods cannot measure the quality of dialogue act sequences produced by the user simulation model. Dialogue-level evaluation methods identified by the authors are:

- Task completion: κ coefficient (based on a confusion matrix), number of turns (task completion time), ratio of successful dialogues. The authors believe each of these measures is indirect and none can be used alone.
- Perplexity, Log likelihood: scalar values that can be used for ranking, but not for *generalisation* since they measure how well a model predicts data (and therefore penalise unseen items).
- HMM similarity: KL dissimilarity between an HMM trained on real data and one trained on artificial data produced by the first. This model can account for unseen dialogues but cannot adapt to new dialogue policies.
- Cramer-von Mises divergence: does not need to know the distributions, represents how well a user simulation behaves with respect to real users. Unseen dialogues do not affect this metric. It can predict the system's performance but does not compute dialogue similarity (meaning that high scores do not mean realistic dialogues).
- BLEU, Discourse BLEU: Taken from machine translation community, it measures the similarity of semantically equivalent sentences. This metric is able to compute how realistic a generated sentence is. D-BLEU is able to compute

how realistic a dialogue is. This method cannot predict the simulation model's *generalization* as well.

- SUPER: Is a combination of several other metrics. Its goal is to compute how realistic produced dialogues are and their variety. This metric has similarities with Word Error Rate. It cannot predict the system's performance though.
- Human evaluation: Use human judges to evaluate the user simulation model. It is similar to PARADISE [99].
- Absolute performance: Expected cumulative rewards (or return), when the system has been trained using RL. Requires testing with real users.
- Strategy evaluation: Again for systems trained with RL. Compares learnt policies with known already policies and for this it requires a similarity measure. The metric then computes the correlation coefficient between $Sim(\pi^*, \pi_{old})$. A high value means the model behaved well (behaved as a real user would have).

It should be noted here, that De Vault et al. [130] also propose an automated dialogue policy evaluation framework, based on human critics (external referees). Pietquin and Hastie [129] conclude that Cramer-von Mises and SUPER score have most of the desired features they proposed and advise to use KL divergence on advanced n-gram models. This metric will not penalise unseen dialogues. The authors also propose to use IRL to learn the underlying reward function that (real) users try to maximise (their satisfaction). They argue that a user simulation model should change its behaviour when the SDS changes policy, to reflect the real users' efforts to maximise their satisfaction.

1.3 Discussion

Having presented relevant related work, in the following Chapters if this thesis, we present our work towards a robotic assistant targeted for ALE, able to talk with

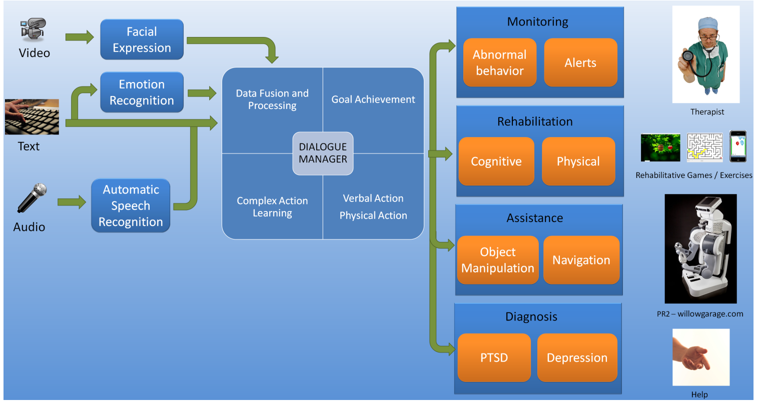


Figure 1.2. Proposed architecture for robotic assistant [1].

the user, understand the user's intentions and act accordingly, either by responding or by acting on the environment. The robotic assistant will be controlled by a state of the art ADS, and will be able to detect behavioural cues and assess severity levels of disorders (such as Post Traumatic Stress Disorder). Figure 1.2 depicts the proposed architecture [1].

To conclude, we propose a Hierarchical Adaptive Dialogue System, able to express complex behaviour, by learning how to solve complicated tasks and how to achieve its goals. In the next Chapter, we present our contribution to Assistive Living technologies; in Chapter 3 we describe our extensive evaluation of RL techniques on DM; in Chapter 4 we present our contributions to ADS; in Chapter 5 we present some applications of the proposed system; in Chapter 6 we present in detail our Post Traumatic Stress Disorder assessment ADS and in Chapter 7 we conclude and present our plans for future work.

CHAPTER 2

Assistive Living Technologies

In this chapter, our contributions to the field of ALE are presented. We focus mainly on two aspects, dynamic optimal sensor placement, in order to position mobile sensors (such as robotic assistants) in real-time in response to changing needs and on recommender systems, specifically designed to operate in ALE. Both aspects can be regarded as integral parts of a robotic assistant, which is our overarching goal and can also be applied to many other applications.

2.1 Sensor Placement in Assistive Living Environments

Sensors play a very important role in ALE, as this is how the system receives input and makes inference on the various aspects of the user's life. Placing sensors optimally, thus, is of great importance, as it can lead to increased coverage of the space, reduced costs and better response to critical situations or environmental changes in general. To address this issue, we proposed an extension of the Max Sum Decentralised Coordination (MSDC) algorithm [46], presented in the next Section, named Extended MSDC (EMSDC) [131]. Our aim was, for a given environment layout, to find a placement solution for multi-tasking static sensors, while also allowing mobile multi-tasking sensors in the environment.

In our model, sensors belong in three categories: static sensors; mobile sensors that we can control, such as sensing robots; and mobile sensors that we cannot control, such as wearable sensors. Sensors can also be characterised according to the number of tasks they can perform (whether simultaneously or switching between tasks): single-tasking sensors, such as microphones; and multi-tasking sensors, such as audio-visual (cameras), Microsoft's kinect, robots and other, custom-built, sensors.

We address the placement problem by utilising a decentralised message passing framework, where each sensor is modelled by an agent in a multi agent system. In short, each agent exchanges location and task preferences, allowing such information to flow through the agent network, resulting in an equilibrium that achieves the overlap and coverage constraints. Each sensor can make decisions based on local information and affect its neighbours' decisions through message passing. MSDC is presented in detail in [46], and we reproduce the main attributes in the following Section for the reader's easy reference.

2.2 Max-Sum Decentralised Coordination Algorithm

In this section we briefly present MSDC, where sensors are modelled as agents, which are in composed of a variable and a utility function. More specifically, utility functions are represented as factors, and task/location of each agent is represented as a variable. The agent network, thus, forms a factor graph. We now describe the messages exchanged between one agent's variable and another agent's factor, and vice versa. The interested reader is referred to [46] for a full description of the algorithm.

Initialisation: Each variable node is randomly initialized, as are all outgoing messages. According to [46], the algorithm is guaranteed to converge to a solution.

Variable to Factor Message: Messages from a variable n to a factor m are computed by calculating the sum of incoming messages from all neighbouring factors except for factor m, plus a normalisation constant α. The message is calculated as follows:

$$Q_{n\to m}(x_n) = \alpha_{nm} + \sum_{m' \in ne(n)\setminus m} R_{m'\to n}(x_n) \tag{2.1}$$

α_{nm} is a scalar such that:

$$\sum_{x_n} Q_{n\to m}(x_n) = 0 \tag{2.2}$$

Factor to Variable Message: Messages from a factor m to a variable n are computed essentially by a local blind search combined with information from incoming messages from the rest of the graph. To calculate the message, we need to find a combination of states for m's neighbouring variables, except for n, that maximize the utility (i.e. the factor's value) plus the sum of all incoming messages except the one that came from n. The message is calculated as follows:

$$R_{m\to n}(x_n) = \max_{X_{m\setminus n}}(U(X_m) + \sum_{n'\in ne(m)\setminus n} Q_{n'\to m}(x_{n'})) \tag{2.3}$$

where X_m is a set of variables that are the neighbours of m.

Marginal: To compute the probability that a variable x is in a specific state, we have to sum all incoming messages to x:

$$Z_n(x_n) = \sum_{m\in ne(n)} R_{m\to n}(x_n) \tag{2.4}$$

In acyclic factor graphs these Z_n represent a solution to the utility maximization problem (or social welfare maximization problem). If however the factor graph contains cycles, Z_n represent an approximate solution:

$$Z_n \approx \max_{X_{m\setminus n}} \sum_{m=1}^{M} U_m(X_m) \tag{2.5}$$

To compute the optimal state for a variable x, we simply have to calculate:

$$\arg\max_{x_n} Z_n(x_n) \tag{2.6}$$

In order to gain better intuition of the meaning of the exchanged messages, we present a non-mathematical explanation:

- When a factor M sends an R message to a variable N, it is as if the factor says: "N, I prefer that you are in state S".

- When a variable N sends a Q message to a factor M, it is as if the variable says: "M, my neighbours prefer my state to be T".

- When a variable n calculates Z, it is as if the variable says: "I am choosing a state based on which state my neighbours mostly prefer for me".

2.3 The Extended Max-Sum Decentralised Coordination Algorithm

To deal with the problem of optimal placement, we created an extended version of the Max-Sum Decentralised Coordination (MSDC) algorithm. This version, unlike MSDC, takes into account not only the state of the agents but also the location of the agents (represented by utility - factor pairs). The main idea is that each agent has two types of states, task and location. This means that instead of having one factor graph, we have two, where at the second one variables represent agents' locations and utilities measure how good these locations are for each agent (typically a measure of the overlap with its neighbours multiplied by a Gaussian function). We then run the MSDC two times, once for each factor graph, i.e. once for task selection and once for placement. As is the case with task selection, the agents exchange preferences on each other's location instead of their own actual location. Each agent then tries to push its neighbours away, to the direction that maximises each neighbour's utility. MSDC's performance has already been proven in [46]. Our algorithm's running time is twice the running time of MSDC, but asymptotically the complexities are the same.

The agents exchange messages in order to maximise that utility. The extra messages exchanged are:

From Factor to Variable:

$$T_{m\rightarrow n}(x_n) = \max_{d\in D}(U_m(p_n, d)) \tag{2.7}$$

where D are all possible directions (e.g. Up, Down, Left, Right) and p_n is the location of the n_{th} agent.

Marginal: The marginal is similar to $Z_n(x_n)$:

$$W_n(p_n) = \sum_{m\in M(n)} T_{m\rightarrow n}(p_n) \tag{2.8}$$

To compute the optimal position for a variable p, we calculate:

$$\arg\max_{p_n} W_n(p_n) \tag{2.9}$$

In practice, however, agents exchange Q messages as defined in [46] and a unified $RT_{m\rightarrow n}(x_n, p_n)$ message:

$$RT_{m\rightarrow n}(x_n, p_n) = [R_{m\rightarrow n}(x_n), T_{m\rightarrow n}(p_n)] \tag{2.10}$$

The marginal becomes:

$$ZW_n(x_n, p_n) = [Z_n(z_n), W_n(p_n)] \tag{2.11}$$

Each agent then computes its optimal state and position by maximising:

$$\arg\max_{x_n, p_n} ZW_n(x_n, p_n) \tag{2.12}$$

which means that finally each agent selects the pair (state, location) with the greatest gain. The distance that the agents will move towards their selected direction is a tuneable parameter of the algorithm.

2.3.1 Experimental Setup

We evaluated several aspects of the proposed algorithm, such as optimal placement of single or multi tasking agents, fault tolerance, and response to environmental changes. For our evaluation we made the assumption that we have a relatively large area to cover (compared to the agents' ranges) and that each sensor's range is circular. Note that these assumptions were alleviated in later work [132]. Critical areas are represented by a three dimensional Gaussian Map. We used the graph colouring metric (each task is modeled as a colour) to solve the optimal placement of multi tasking agents, to make sure that agents do not overlap (w.r.t the task each sensor performs). It should be noted here that sensors can alternate tasks as circumstances change.

We use the graph colouring utility from [46]:

$$U_m(X_m) = \gamma_m(x_m) - \sum_{i \in ne(m)} \sum_{j \in C(i,m)} x_i \oplus x_j \tag{2.13}$$

where $x_i \oplus x_j$ is 1 if $x_i = x_j$ and 0 otherwise, and C is defined as:

$$C(i,m) = \{k \in ne(m) | k > i \wedge (i \in ne(k) \vee k \in ne(i))\} \tag{2.14}$$

For the placement part we used the following utility:

$$U'_m(X_m) = \sum_{i \in ne(m)} \sqrt{(p_m - p_i)^2} \tag{2.15}$$

where p_i is the position of the i_{th} agent.

When we have a Gaussian Map we simply multiply U'_m with the Gaussian:

$$U'_m(X_m) = \sum_{i \in ne(m)} (\sqrt{(p_m - p_i)^2} G(p_i)) \tag{2.16}$$

In order to calculate the total coverage, we need to first calculate the total overlap between the agents. The calculation of the overlap however can be very hard in the general case. We tackle this problem using an approximate Monte Carlo technique. We randomly choose 50,000 points, which is like throwing 50,000 darts at the area to cover, and we count the number of hits for each dart. If there are no hits that point is not covered, whereas, if there are more than one, there is overlap at that point. We can then measure the hits per dart and name that ratio *reward*. We increment the *reward* by one for each dart with exactly one hit, and by $\frac{1}{hits}$ if there are more that one hits. The optimal would be 1 hit per dart, thus $optimal = 50,000$.

Another way would be to increment *reward* by one for each dart with more than one hit, and then use the following formula to calculate the area covered by the agents (which is $\sum_{i=1}^{N} \pi\rho_i^2 - overlap$):

$$area = \frac{\frac{reward}{optimal} areaToCover}{min\{areaToCover, \sum_{i=1}^{N} \pi\rho_i^2\}} \tag{2.17}$$

We consider that a cycle has passed when all agents have had the opportunity to change their state. We run each experiment for 250 cycles. Each agent's range was set to 30 units. The dimensions of the environment are 460 x 980 = 450800 units, so each agent alone can cover 0.6% of the area. Each experiment was run 10 times for each dimension, and we calculated the average.

2.3.2 Results

For our experiments we used the Gaussian map and run the algorithm for 10 to 150 agents. Figure 3 shows the percentage of the covered area versus the number of agents. We can see that the total area covered rises rapidly in the lower dimensions and slower in the higher dimensions. This is because the agents are trying to cover the high interest areas (red) first, leaving others (blue) less covered. This might be a desirable feature, since we may have some overlap in the red areas, but this also means redundancy and increased fault tolerance. It is possible to tune the algorithm

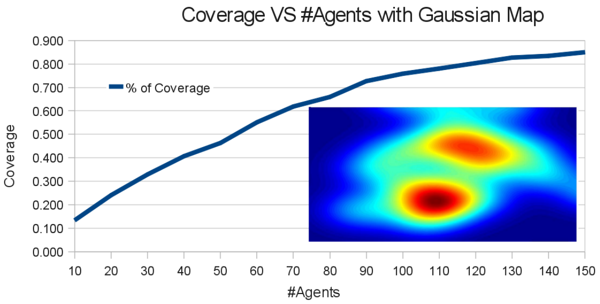


Figure 2.1. Percentage of coverage versus number of agents when using a Gaussian map.

and put more weight on the overlap between the agents and less on the effect of the Gaussian map. This way we will have less overlap in the red areas and the agents will spread more.

2.3.3 System response to environmental changes

An interesting problem is how the system will respond to a change in the environment. To model this, we use two different Gaussian maps, depicted in figure 4, where the second map (bottom image) has one more "important" region. This could be an event like a fire in the kitchen or a person falling in the bathroom. The first image shows the initial random placement of the sensors, before the EMSDC algorithm has been run. The percentage of coverage in this case is 39.1% The second (middle) image shows the coverage of the area after the execution of the EMSDC algorithm but before the occurrence of the critical event. The coverage in that case is 68.5%. We then changed the map, at which point the coverage suddenly became 60.7% since an important area was not covered. After the algorithm ran for 100 cycles the new resulting coverage increased to 64.2%. Figure 2.2 - bottom, shows the final position of the agents. We can clearly see that the agents adapt very well to the change in their environment. Note that it is not possible to achieve the initial percentage of coverage

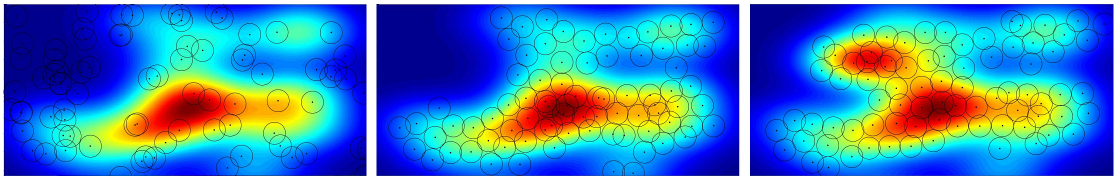


Figure 2.2. The top image shows a random starting state of the system when using a Gaussian map. The middle image shows the state of the system after the execution of the EMSDC algorithm. The bottom image shows the final state of the system after a change in the Gaussian map has taken place and the system has converged to a new solution.

with the same number of sensors, since after the map change there is a bigger amount of "important" regions to be covered.

2.3.4 Fault Tolerance

The two main benefits of using multi agent systems are decentralised control, meaning that each agent performs small tasks that can be performed by low cost devices, and fault tolerance. Here we prove that EMSDC performs very well in the presence of failures. To test the fault tolerance of the system, we compared it to a static system, i.e. a system where the sensors cannot move to compensate for failures. We run EMSDC using 100 agents, calculated the coverage after 0% to 40% agents have failed randomly and compared the results with the coverage of EMSDC in the presence of failures. To simulate failures, we use a model where each sensor has a probability p to fail at each cycle. After that point the sensor becomes useless either because it cannot take measurements or because it cannot transmit/receive data. We can see the results in figure 2.3. The very small decrease in coverage for EMSDC means that as the agents in important sections fail, others come and take their place. Contrary if the placement of the sensors could not be re-organized after some sensors have failed, important regions could remain uncovered and that would result in a

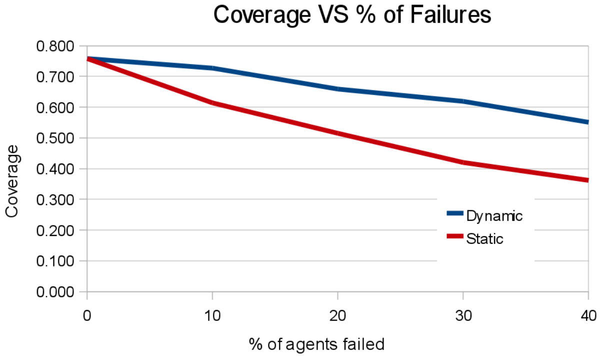


Figure 2.3. Percentage of coverage versus percentage of agents failed, for a static and dynamic system. We can see that the dynamic system adapts well to failures.

rapid decrease in percentage of coverage. As we can see from the graph, with 10% of sensors failed we already have a 13% difference in the coverage between the static approach and our dynamic system.

Although not shown here, the system is also very tolerant to lossy communication, since messages sent at time t are not very different from messages sent at time $t-1$. Each agent stores in memory the received Q and R messages, so at each cycle, if an incoming message is lost the agent will use the previously stored one. Also, messages are exchanged constantly and rapidly, so the agent will most likely receive within reasonable time an updated message coming through a lossy link. Lossy communication can slow down the convergence of the system into an optimal state but does not significantly affect the final outcome.

So far we have discussed optimal placement of single-tasking agents. A multi tasking agent is an agent that can perform more than one tasks, e.g. capturing of video, sound and temperature. It is not necessary that all agents are capable of performing the exact same tasks or that the agents can perform multiple tasks simultaneously or one at a time. Generalising our model to handle multi-tasking agents is straightforward, as we only need to do the following. Assume that we have

K unique tasks the agents are capable of performing (in the general case, each agent can perform a subset of K). We can then apply MSDC with each agent having 2^K states, where each state, converted to a binary number, represents the tasks that are "on" or "off", depending on whether the corresponding bit is 1 or 0.

The effect of EMSDC on our system is that we can derive patrolling routs for mobile robots, including our robot assistant, as well as raise alerts when critical events occur, that our robot can intercept and navigate to the critical area. Continuing our work on sensor placement in ALE, we developed a web-based tool that employs EMSDC and provides various features [132]. It is targeted at the designer or supporting technician of an ALE rather than the users themselves and is presented in the following Section. A description of another application of EMSDC for assistive robot placement can be found in [133].

2.4 EMSDC Web Tool

In this Section, we present a web-based tool that we developed, which applies EMSDC. This tool is part of a larger system (zScope) and is targeted to the technician responsible for installing and maintaining the sensor network. Our tool has two main modes, *Drawing* mode and *Monitoring* mode. In the first mode, also depicted in Figure 2.4(a), the user designs the layout of the space to be monitored. We provide functionality to define walls, tweaking their length and thickness and also define critical areas. After the design is complete, the user defines the type and number of sensors to be installed, and the system runs EMSDC to propose an installation solution. The user then has the option to fine tune the placement solution, to account for unforeseen issues, such as the inability to install a sensor due to physical constraints. When the placement phase (Figure 2.4(b)) is complete, the system goes into *Monitoring* mode (Figure 2.4(c)), where it constantly checks for faulty sensors. The system can issue warnings and alerts, depending on the health of the sensor network, and if enough sensors have failed to have a significant impact on system coverage, the system runs EMSDC again, attempting to reconfigure the healthy sensors and achieve as much coverage as possible. The new solution is then presented to the technician, who can decide whether it makes sense to relocate several sensors or replace the faulty ones. Our approach can have a great impact on cost, as the technician may not actually need to replace every faulty sensor, or, in an emergency, the reconfiguration of existing sensors may be much faster than purchasing replacements. Last, for research and evaluation purposes, the tool can go in a *Simulation* mode (Figure 2.4(d)), where failures and system responses are simulated.

The tool is able to recommend sensor placements and the user can adapt that recommendation to account for unforeseen issues (power cables, surfaces where we cannot mount specific sensors etc). After the placement is complete the tool goes into System Monitoring Mode and monitors the devices for any failures. When a failure occurs the system can respond in several ways:

- Adjust the recommended sensor placement to compensate for the failures and increase coverage as much as possible.
- Move mobile sensors to compensate for the lack of coverage. We may have some robots standing by.
- Raise an alert to the technician.
- Raise a warning to the technician.

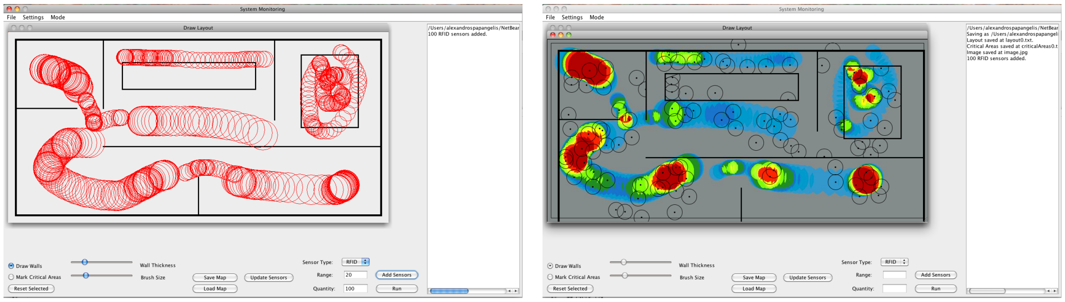
The proposed tool has two modes. The *Environment Design* mode, where the user designs the environment layout, selects the sensor set and defines critical areas for each sensor type. The system can then recommend near optimal positions for each sensor, using a Max Sum Message Passing algorithm, as discussed in Section 2.3. From the user defined critical areas we are able to create a 3-dimensional map (similar to the Gaussian maps of [1]) that will be the utility function $U(X_m)$. The only problem with this approach is that different types of sensors (eg smoke sensor at the kitchen, motion sensor at door frames) might have different critical areas, so we ask the user to define them separately and run the algorithm for every sensor type. In the future we will have a heuristic that will combine all critical areas and the algorithm will run only once.

2.4.1 Environment Design

The tool provides a way of drawing an environment layout and specifying the critical areas. The user can draw walls and adjust their thickness from the corresponding slider. The user can also define the critical areas by using the brush to paint the area of interest, as depicted in Figure 1 - top. Brush intensity and size can be adjusted. There is also an option for brush smoothing where higher values will be assigned towards the center of the brush and lower as we move away. The user can also load a layout and the corresponding critical areas from a file.

From the defined critical areas we create a 3-dimensional map that corresponds to agent rewards (Gaussian Maps in [1]) and will be used in the automatic sensor placement process. During this process, in short, the sensors will try to cover the critical areas as much as possible and reduce the overlap as much as possible (this is a tunable parameter of the algorithm [1]). This is achieved through the Extended Max Sum Decentralised Coordination algorithm [1].

There is also a map smoothing option, where we find the peak of the 3-dimensional map and create a small inclination towards it according to the following formula:

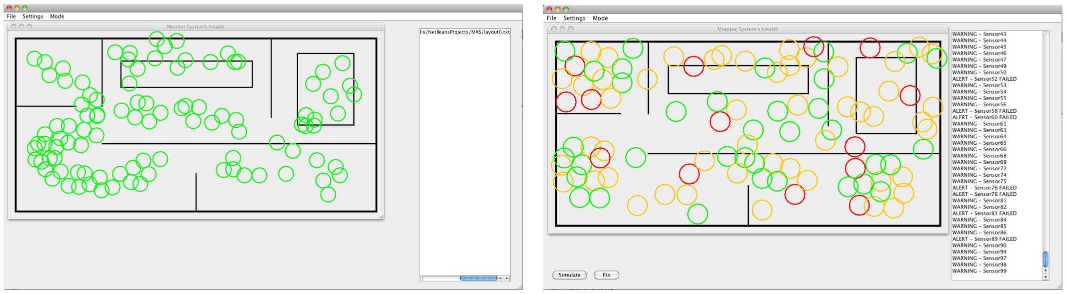


Sensor Type: RFID
Range: 20
Quantity: 100

```
/Users/alexandrospapangelis/NetBear
100 RFID sensors added.
```

```
/Users/alexandrospapangelis/NetBear
Saving as /Users/alexandrospapangeli
Layout saved at layout0.txt.
Critical Areas saved at criticalAreas0.t
Image saved at image.jpg
100 RFID sensors added.
```

(a) Layout Drawing (b) Sensor Placement



```
lis/NetBeansProjects/MAS/layout0.txt
```

```
WARNING - Sensor43
WARNING - Sensor44
WARNING - Sensor45
WARNING - Sensor46
WARNING - Sensor47
WARNING - Sensor49
WARNING - Sensor50
ALERT - Sensor52 FAILED
WARNING - Sensor53
WARNING - Sensor54
WARNING - Sensor55
WARNING - Sensor56
ALERT - Sensor58 FAILED
ALERT - Sensor60 FAILED
WARNING - Sensor61
WARNING - Sensor63
WARNING - Sensor64
WARNING - Sensor65
WARNING - Sensor66
WARNING - Sensor68
WARNING - Sensor69
WARNING - Sensor72
WARNING - Sensor74
WARNING - Sensor75
ALERT - Sensor76 FAILED
ALERT - Sensor78 FAILED
WARNING - Sensor81
WARNING - Sensor82
ALERT - Sensor83 FAILED
WARNING - Sensor84
WARNING - Sensor85
WARNING - Sensor86
ALERT - Sensor89 FAILED
WARNING - Sensor90
WARNING - Sensor94
WARNING - Sensor97
WARNING - Sensor98
WARNING - Sensor99
```

(c) System Monitoring (d) Monitoring Simulation

Figure 2.4. The various modes of operation of our EMSDC web-based tool.

$$map_{x,y} = \frac{smoothValue}{\sqrt{(x - x_{MAX})^2 + (y - y_{MAX})^2}} \tag{2.18}$$

where (x_{MAX}, y_{MAX}) are the coordinates of the highest peak in the map and $smoothValue \in [0, 1]$, selected by the user.

This does not work if we have several peaks, and the user should exercise caution when selecting this option. In the final version of the tool, however, this is not expected to be a problem. This option is provided to eliminate flat areas in the maps, where the sensors have no incentive to move. The maps can be saved using either a default name or one defined by the user. When the user has finished drawing the layout and defining the critical areas and the sensor set he can either place the sensors himself by pressing the *Update Sensors* button or let the system recommend a placement by clicking the *Run* button, as depicted in Figure 2.4(c). Automatic sensor placement is done by Extended Max Sum Decentralised Coordination [1] algorithm, which was described in section 3.3. After the placement is complete, either automatically or manually, the user may select to switch to *Monitoring Mode* from the *Mode* menu.

2.4.2 System Monitoring

In this mode critical areas are not visible and the user can only see the environment layout and the sensors in their final positions, as depicted in Figure 2.5 - top. From the *Settings* menu the user may choose how he wishes to receive alerts, for example via email, via message boxes or appearing in the text box. Alerts and warnings can also be deactivated, but will appear in the system's log.

To monitor the sensors' health the system checks to see if all sensors are active, at regular time intervals. When the system fails to receive signal from a sensor, that sensor's colour will turn yellow. If the system does not receive signal from that sensor for a predefined period of time that sensor will be considered to have failed, its colour will turn red. If a sensor is inactive or has failed, a warning or alert is raised accordingly. Before raising an alert the system attempts to automatically provide a solution, i.e. run the placement algorithm again excluding the failed/inactive sensors. This can be used as a recommendation in the event where we immediately need to restore coverage and repairs might be time consuming.

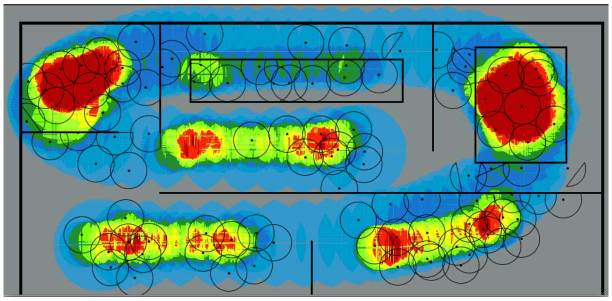
2.4.3 Evaluation

The system was evaluated using real data from the HERACLEIA Assistive Living apartment and also using synthetic data. Real data came from SunSPOT sensors and Pressure Mats installed in the HERACLEIA apartment and were used mainly to evaluate *System Monitoring* Mode. We measured the system's time to respond (detect a failure, raise an alert or warning, recommend new placement) and the percentage of failed sensors it was able to detect. The system's average response time was 3.22 sec and the system was able to detect 100% of sensor failures.

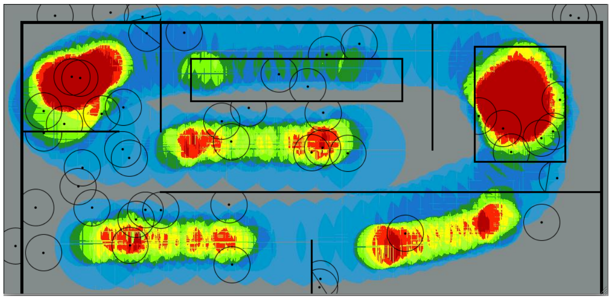
To evaluate sensor placement (which includes sensor re-positioning when planning a new placement after a failure) we used simulations, as depicted in Figure 2.4(d). We modelled two types of failures, signal and hardware failures and incorporated this model in each sensor. The system has no knowledge of the type of failure and has a tolerance parameter, measured in time steps, to account for signal failures. If a sensor fails for more than the predefined time step threshold it is considered to have a hardware failure. Signal failures are depicted with yellow (warnings) and hardware failures with red (alerts). We measured the system's total coverage percentage before a failure and after the system responded and re-positioned the sensors. Results of sensor re-positioning (i.e. system's response) are very promising and clearly show a very small loss in coverage of critical areas, as we can see in Figure 2.5 where 50 % of 100 sensors fail.

The proposed tool is able to provide robust recommendations on sensor placement and monitor the system's health. It raises alerts and warnings but it first attempts to solve the problem automatically. It is web based, easy to use and can be easily extended to provide data analysis and inference on human behaviour.

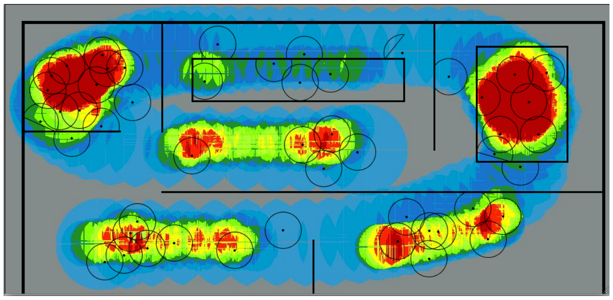
We plan to integrate this tool into a web interface called zScope. zScope's purpose is to monitor human behaviour and analyze monitoring data in order to provide more interesting information. A straightforward extension for this tool therefore is to collect data from the sensors (not just monitor system's health) and perform behavioural analysis using various Machine Learning techniques. Another extension is to take into account multi tasking sensors and find a heuristic that creates a common critical area map which could be a linear combination of all critical area maps.



(a) Steady state



(b) Re-positioning phase



(c) After response

Figure 2.5. The three phases of the proposed System Monitoring Tool.

Last, we plan to incorporate battery and time constraints, add a feature to optimise cost of devices against total coverage and perform extensive evaluation using real and simulated data.

2.5 Recommender System for Assistive Living Environments

In this section, we present a recommender system that was designed to take into account the special needs of ALE users, such as the fact that they may not be able to communicate keywords of items they are interested in [134]. The main idea of our approach is to apply Natural Language Processing (NLP) methods and weighted tagging of keywords, to label, rate, cluster and finally recommend items of interest to the user. Clustering occurs in an online hierarchical way. Our system also provides functionality to extract tags and ratings automatically from descriptions of items, provided by the users.

2.5.1 Introduction to Recommender Systems

Recommender Systems have been widely used over the past years trying to help users navigate through the overwhelming amount of information available. A good Recommender System can prove very profitable in businesses such as www.amazon.com or www.net flix.com by recommending products the users can buy (or products that other users have bought to take advantage of *peer pressure*). The greatest challenge for a Recommender System is to recommend items from clusters that the user has never seen and might like. Apart from these known advantages, Recommender Systems are also very useful in Assistive Environments. In this setting users normally are less able than typical computer users, either physically or mentally. In any case their ability to efficiently search for information is limited and here is where a Recommender System can prove extremely helpful. Recommender Systems targeted for Assistive Environments therefore should take this user inability into account, be more flexible and provide more natural and intuitive ways of using them, specifically at the point where users are describing what they are looking for. We propose a *content based* Recommender System that uses Natural Language Processing techniques, such as Automatic Summarization or Sentiment Analysis, to analyze (automatically tag and rate) and dynamically cluster items. Motivation to build our system came from the fact that using tags, ratings or historical data is sometimes not enough for a Recommender System and more than this, in Assistive Environments typical users are less efficient in searching for items using pre-defined tags. It is a rather mathematical approach that fails to capture the meaning of tags or item descriptions. In order to correctly identify the meaning, we need to take into account the context in which it was expressed. *Context*, as defined in [7], is information that describes an entity's situation or environment. *Context Aware* is any application that exploits *context*. In this work, the meaning of context awareness is two fold. First it means understanding item descriptions by assuming tags are not independent of each other, and second it means recommending items to users taking into account their environment, which could be their item browsing history, their website browsing history, the contents and opinions in a forum they are participating in and many others, depending mainly on the application. All these could be defined as *current context*, versus *long term context* that would be predefined user preferences. In this work however we only use *current context*. Apart from Assistive Environments our system could easily be used for database population by automatically tagging and rating items from reviews or descriptions found on the web, for advertising in chat rooms or forums or an application that uses browsing history, chat history (with a focus on questions) etc to create a query through Natural Language Generation techniques and then ask the recommender system for similar items. Last, the proposed system does not store users' ratings for each item, but rather a collective "opinion" which is updated with each new rating or description of the item. Although there has been substantial work in Recommender Systems, there has been little to no work in adapting a Recommender System to meet Assistive Environment requirements. These requirements include the ability of a system to provide companionship and to understand a user that is not trained to use technology. Our system not only applies novel algorithms for recommendation tagging and rating but is also targeted for assistive environments.

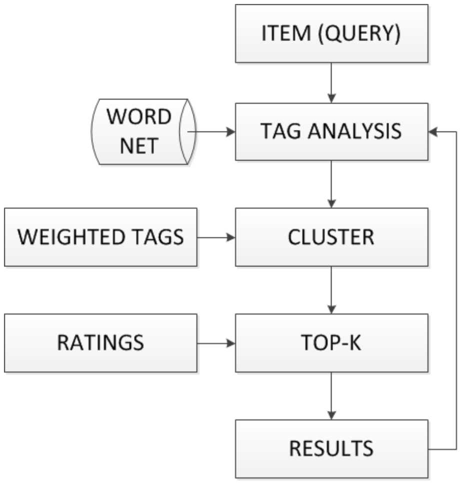


Figure 2.6. The proposed architecture for item recommendation..

Through natural language understanding and semantic analysis techniques it is able to understand what the user exactly needs. The user does not need to provide tags, just a description in natural language of what he is looking for.

2.5.2 Related Work

In this section we will present a brief overview of recent work that is related to ours, focusing on Recommender Systems since there is no work on Recommender Systems in Assistive Environments. In [6] the authors propose a wearable context aware assistant that is able to adapt to user context. Context is defined here as activity, location, identity and time. [5] evaluates a model that personalizes the order in which news articles are presented to the user according to his long term interest profile or according to the current semantic context of interest or both, using *News@hand* Recommender System [11]. [4] proposes a context aware personal assistant (*Personal Digital Secretary*) that combines user modelling and context awareness techniques. In [3] the authors propose a context aware Recommender System that exploits the semantic web. [2] presents a context aware recommender system that deploys personalized services (such as context discovery service, contextualization service and others) to provide context aware recommendation. The authors use both user context and preferences to make recommendations. [1] is an interesting and effective approach to the difficult problem of recommending citations for a given article, at specific locations in that article (*citation placeholders*). In [10] the authors propose a new algorithm for content based image retrieval that uses dynamic clustering, and their aim is to eliminate irrelevant images from the results, as is the case in many image retrieval systems. [9] presents techniques that incorporate user context (location, time, affiliation) in information requests such as request expansion, ordering as a feature of query language, context templates as an automation mechanism, combining contextual attributes. [8] takes into account user context, in the sense that a user might be in a different mood at different times of day, in order to recommend music.

2.5.3 A Recommender System for ALE

In this section we will present our novel *content-based* Recommender System, which offers an easy and intuitive way for item tagging and rating, without using historical data as most traditional Recommender Systems do. A user can describe an item (including his opinion) as he would to his friends. Users of our system can tag an item using any word they like and the system will convert them to either tags or ratings. They can also provide a short description of the item instead and the system will extract tags via Automatic Summarization techniques and ratings via Sentiment Analysis. Last, we provide the option of explicit item rating. The system then uses inferred and explicit tags and ratings to select the *top-k* items that will be recommended to the user. Figures 2.6 and 2.7 describe our proposed system. Figure

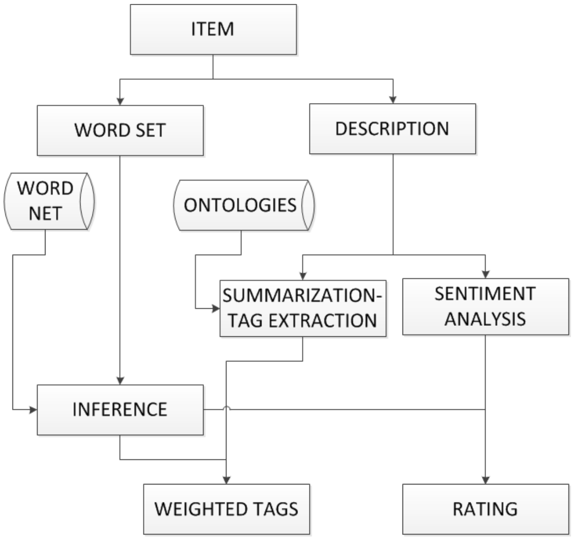


Figure 2.7. The proposed architecture for weighted tagging and rating.

1 depicts the tagging and rating process and Figure 2.6 depicts the recommendation (and dynamic clustering) process.

2.5.3.1 WordNet

WordNet is a publicly available database for the English language that was first developed by Princeton University in 1985 (wordnet.princeton.edu). It contains adjectives nouns and verbs grouped into synonyms called *synsets*. These *synsets* are linked together by lexical or semantic relations and form a network of conceptually related words. WordNet is a very useful tool for natural language processing tasks and has been used extensively in research.

2.5.3.2 Weighted Tagging

Each item in the Database is represented by a vector where each dimension corresponds to a tag. The value of each dimension reflects the ratio of users that have used this specific tag to describe the item. There are two ways to tag an item, either explicitly or through the item's description. While the first case is straightforward, the second requires some analysis. To achieve automatic tag extraction, we first get words that refer to the item (assuming we know or are able to infer the item's name). An example tool for this is *www.opencalais.com* that offers Named Entity Recognition. We also use Automatic Statistical Summarization techniques to get the most important words of the text, and through some final processing using Natural Language Understanding techniques we are able to extract words (tags) that describe the item. Note here that the aforementioned techniques are able to distinguish between verbs, adjectives etc. We also try to infer the WordNet sense (meaning) of each tag based on the other tags in the list. If we cannot get an answer with enough confidence we do not resolve the sense of the tag and leave it for further analysis at the recommendation stage (if results are not satisfactory we will cluster tags based on other senses through dynamic clustering). Algorithm 1 presents the *Tag Sense Inference* algorithm. Note here that we get the senses of each tag by asking WordNet. The algorithm receives as input a tag t and a list of tags, which is all other tags used by the user. We then get all unique synonyms of each sense of each tag in the list, as well as all synonyms of all senses of t. We assign t the sense for which it has the highest number of synonyms with the tags in the list. If there is a tie the most frequently used sense wins (it will be higher in the list returned by WordNet). *Enough confidence* roughly means a threshold on *Count*.

2.5.3.3 Rating

The proposed system offers three ways for a user to rate an item. First by providing an explicit rating score (for example 3 out of 5 stars), second by tagging it with words such as "fantastic", "terrible" etc, and last by providing an opinion (either explicitly or implied) in the item description text. Through Natural Language Un-

Algorithm 1: Tag Sense Inference(*tag, tagList*)

```
$tagListSynonyms \leftarrow Empty$
for all tags in tagList do
    $tagListSynonyms \leftarrow WNSynonyms(tag)$
end for
$tagSynonyms \leftarrow WNSynonyms(tag)$
for all synonym of tag do
    Count matching synonyms with $tagListSynonyms$
end for
if Sense found with enough confidence then
    Assign sense corresponding to maximum count
else
    Return 0
end if
```

derstanding techniques whenever we find a tag that expresses emotion/opinion such as ”fantastic” we convert it to a rating score. This score is then used to calculate the item’s average rating. Through Sentiment Analysis and Automatic Summarization techniques it is possible to extract rating scores (showing how positive or negative the author’s opinion is) and tags respectively from the description of the item. It is also possible to calculate a general consensus by summarizing all descriptions and analyzing the sentiment. The whole process of tagging and rating an item is depicted in Figure 2.7.

Algorithm 2: Item Sense Graph(*weightedTagVector*)

```
for all tags in weightedTagVector do
    $hypernyms \leftarrow WNHypernyms(tag, sense)$
end for
Combine all hypernyms graphs to a single hypGraph
Return hypGraph
```

2.5.3.4 Clustering

Our system dynamically clusters the items enabling user (or query) specific recommendation. This is achieved by structuring abstract senses of tags in a hierarchical fashion and then selecting the level of abstraction we need for clustering. Before we can talk about clustering though, we need to define some similarity measures. Tag similarity is computed by counting how many synonyms two tags have in common according to WordNet. Weighted tag vector similarity is computed by calculating the Pearson correlation score on the common dimensions of the two vectors. We might also need to penalize vectors that have too few dimensions in common, but more experiments are needed to support this. A word in WordNet can have several senses (meanings) and according to those senses it can have different synonyms. We can take advantage of this fact to find other synonyms of a tag when the recommendation results are not satisfactory to the user. Word sense disambiguation however is not explored in this work but will be part of our future work. In order to enable dynamic hierarchical clustering we need to be able to somehow group tags in clusters that correspond to a concept (more abstract than the tags that belong to that cluster). This is captured by a Sense Graph, which is a layered graph where each layer corresponds to an abstraction level, as defined by WordNet. Nodes in this graph are abstract concepts and leaves are tags.

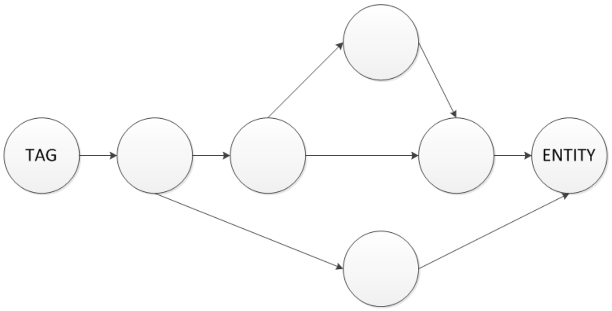


Figure 2.8. Sense Graph of a tag. The tag is at the bottom leaf of the graph and the most abstract sense (Entity in most cases) is the root.

To generate the Sense Graph of an item, we get the hypernyms for each tag through WordNet and represent them as a small graph (it is a graph since children can have many parents), as depicted in Figure 2.9. These graphs always have a root, and that would be the node *Entity*, and a single leaf node (or tail) which would be the tag itself. As we move towards the root of the graph we find more abstract senses such as *Physical Entity* or *Measure*, while towards the tail we find concepts closer to the tag. We then combine these graphs into one, eliminating any duplicate nodes. This combined graph represents an item. The weight of a non-leaf node is the sum of weights of its children. We then combine these Sense Graphs to a single Sense Graph that represents the whole Database (Figure 4). This is a one time processing step and once completed we only need to update the graph with any new item rating/tagging. After having created the Sense Graph we can cluster the items by ”cutting” the graph at an abstraction level of our choice (a form of feature selection) and if the results are not satisfactory we can move to lower or higher levels of the graph. Generally lower levels provide more detail (more clusters of fewer items each) and higher levels provide less detail (fewer clusters with more items each). Algorithm 2 presents the generation of the Sense Graph for a single item and Algorithm 4 presents the combination of many Sense Graphs into one. Each cluster is represented by a ”mean” vector, the centroid. At each iteration of the recommendation process these centroids need to be updated (the system is re-clustering the items). This centroid, is composed of the unique tags of all weighted vectors that belong to that cluster, and the value at each dimension is the corresponding mean. Algorithm 3 presents how the centroid is calculated.

Algorithm 3: Centroid Calculation(*Cluster*)

```
$Centroid \leftarrow Empty$
$Count[sizeOf[Cluster]] \leftarrow Empty$
for all $weightedVector \in Cluster$ do
  for all $tag \in weightedVector$ do
    if $tag \notin Centroid$ then
      Add $tag$ to $Centroid$
      $Centroid_{tag} \leftarrow 0$
      Add $tag$ to $Count$
    end if
    $Count_{tag} \leftarrow Count_{tag} + 1$
    $Centroid_{tag} \leftarrow \frac{Centroid_{tag} + tagWeight}{Count_{tag}}$
  end for
end for
```

2.5.3.5 Recommendation

The system first recommends the *top-k* most similar items from the cluster where the item of interest belongs. If the user is not satisfied, we re-cluster the items using more detail by *cutting* the Sense Graph at a lower layer. This way smaller and more meaningful (of a "meaning" closer to the query item) clusters are formed and again we recommend the *top-k* items. We sort these *top-k* items in order of decreasing rating scores. The process of dynamic clustering and recommendation is depicted in Figure 1.

Algorithm 4: Sense Graph(*Database*)

```
for all items in Database do
    $hypernymGraph \leftarrow ItemSenseGraph(item)$
end for
Combine all hypernymGraphs to a single Graph
Return Graph
```

2.5.4 Conclusions

We proposed a Recommender System targeted for Assistive Environments that is able to capture the meaning of each item's description, thus providing a more meaningful way to cluster and an intuitive way of searching for items. The system is also able to re-cluster using more information if the results are not satisfactory. This is a novel approach using Natural Language techniques to exploit the several meanings of words used to describe an item.

As future work, we plan to apply deeper linguistic analysis to extract more meaningful information and assess the trade-off between gain in information quality and processing time. A faster approach would be to map tags that the user enters to a pre-defined list of well performing tags, using Natural Language Understanding techniques. Last, we will apply word sense disambiguation techniques to better infer the sense of each tag.

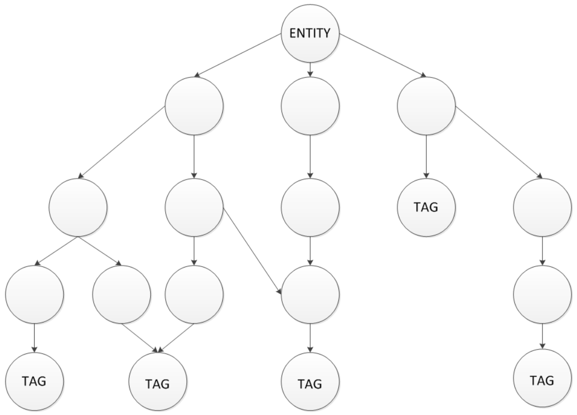


Figure 2.9. The Tag Sense Graph for a small database.

2.6 Discussion

In this Chapter, we have presented our contributions to the state of the art in Assistive Living Technology and, specifically, we proposed a decentralised message passing algorithm able to jointly coordinate agents with respect to the task they are performing and their location, we presented a tool that implements this algorithm using a web interface and last, we proposed a novel content-based recommender system. In the next Chapter, we present some necessary background knowledge in the field of online dialogue management as well as our extensive evaluation of several online reinforcement learning algorithms, applied to that problem.

CHAPTER 3

Online Dialogue Policy Learning

In this Chapter, we present background knowledge regarding online dialogue policy learning. We specifically focus on Reinforcement Learning (RL) and the paradigm we follow to model the dialogue problem. We then present in detail our extensive evaluation of several standard and state of the art online RL methods, our methodology, experimental setup and results.

3.1 Background Knowledge

In this Section, background knowledge, necessary to follow the rest of this work, is presented. Markov Decision Processes (MDP) are presented first, followed by the basics of RL, which is used by the majority of the research community to achieve adaptation in Dialogue Systems. We then proceed to describe the most prominent dialogue management models and how we can apply RL techniques to solve the dialogue problem.

A MDP is defined as a triplet $M = \{X, A, P\}$, where X is a non empty set of states, A is a non empty set of actions and P is a transition probability kernel that assigns probability measures over $X \times \mathbb{R}$ for each state-action pair $(x, a) \in X \times A$. We can also define the state transition probability kernel that, for each triplet $(x_1, a, x_2) \in X \times A \times X$, would give us the probability of moving from state x_1 to state x_2 by taking action a. Each transition from a state to another is associated with an immediate reward, as dictated by the reward function $R : X \times A \to \mathbb{R}$. More specifically, R

is defined as the expected immediate reward after a transition: $R(x, a) = \mathbb{E}[r(x, a)]$, where $r(x, a)$ is the actual immediate reward.

Of interest now is to calculate the cumulative rewards, as collected during the transitions from state to state, until a terminal state is reached. We therefore need to select an action for each state that will (in the future) maximize the rewards. An MDP policy is a mapping that dictates which action to take at each state. An optimal policy is a policy that maximizes the cumulative discounted expected rewards [135]. Semi Markov Decision Processes (SMDP) introduce the concept of continuous time. They allow the system to remain in a state for an arbitrary amount of time, defined by a probability distribution. MDPs and SMDPs, however, make the strong assumption that the system's state is directly observable. This is not always true and this is the problem addressed by Partially Observable Decision Processes (POMDP), where states are replaced by a distributions $b(s)$ (i.e. beliefs) over states.

3.1.1 Reinforcement Learning

Motivation to use RL in the dialogue problem came from the fact that it can easily tackle some of the challenges regarding real-world DS, one of which being error recovery. Hand-crafted error recovery cannot scale, so there is need for automated processes, to learn good error-recovery strategies. More than this, using RL we can automatically learn near optimal dialogue policies and thus maximize user satisfaction. Another important benefit is that such algorithms can be trained using either real or simulated users and continue to learn and adapt with each interaction (online learning). To use RL, we need to model the dialogue system using MDPs, POMDPs or SMDPs. POMDPs have the advantage of modelling uncertainty, and can moreover be transformed to continuous state MDP meaning that the policy optimization problem can be solved with the guarantee that the solution also optimizes the original

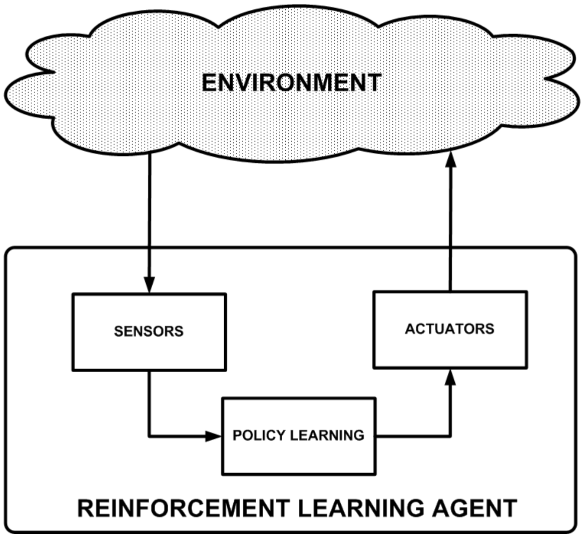


Figure 3.1. A Reinforcement Learning agent, interacting with its environment through actuators and sensors. The policy dictates the agent's actions, while taking into account input from the sensors..

POMDP [115]. SMDPs add temporal abstraction to the model and using them we can, for example model time consuming operations, such as database retrieval.

RL tries to maximize an objective function by learning how to control the actions of a system, as depicted in Figure 3.1, which, in this setting, is typically formulated as an MDP. For every MDP we can define a policy π, which is a mapping from states $s \in S$ and actions $\alpha \in A$ to a distribution $\pi(s, \alpha)$ that represents the probability of taking action α when the system is in state s. This policy dictates the behaviour of the system and the *return* of a policy π is defined as:

$$J^{\pi} = \sum_{t=0}^{\infty} \gamma^t R_t(x_t, \pi(x_t)) \tag{3.1}$$

A policy π is optimal if $J^{\pi}(x) = V^{\pi}(x), \forall x \in X$. To estimate how good a policy is we define the *value function* V:

$$V^{\pi}(x) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | X_0 = x], x \in X \tag{3.2}$$

which yields the expected cumulative rewards when beginning from state x and following policy π, discounted by a factor $\gamma \in [0, 1]$ that represents the importance of future rewards. We can also define the *action-value function* Q:

$$Q^{\pi}(x, \alpha) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t R_{t+1} | X_0 = x, A_0 = \alpha], x \in X, \alpha \in A \tag{3.3}$$

which yields the expected cumulative discounted rewards when beginning from state x and taking action α, again following policy π. Note that $V_{max} = \frac{r_{max}}{1-\gamma}$, where $R(x) \in [r_{min}, r_{max}]$. The goal of RL therefore is to find the optimal policy, which maximizes either of these functions [135].

3.1.1.1 Online Reinforcement Learning

The advantage of online methods for RL is that at each turn we only need to take into account (belief) states that can be reached from the current (belief) state. We thus have to compute the maximum value of the current (belief) state and not for every one. Another advantage is that online algorithms are applicable to dynamic environments as learning never stops and the system is able to adapt to changes.

Many researchers use (inexpensive) approximate offline methods to find upper and lower bounds on the optimal value function and then try to estimate it using an online technique. An example method for estimating the lower bound is to follow the *blind policy*, which selects the same action at every turn. This method is fast but the bound is not very tight. Another way of estimating the bounds is to use point based algorithms such as Point Based Value Iteration (PBVI) [136], Perseus [137] or Heuristic Search Value Iteration (HSVI) [138]. These methods have the advantage of providing tighter lower bounds when we increase the number of sampled belief points (and thus their complexity). Similarly, there are many approaches to estimating the upper bound of the optimal value function, such as using the value function of the underlying MDP or assuming that there is no uncertainty (partial observability) after one turn. A method that can deal with uncertainty is Fast Informed Bound (FIB) [139].

Online reinforcement learning algorithms generally have two stages, the planning (or learning) stage and the execution stage. There are many ways to make an online algorithm faster or more efficient, such as pruning the set of belief states to consider at each turn (e.g. by selecting the most relevant ones) [140].

3.1.1.2 Hierarchical Reinforcement Learning

Hierarchical Reinforcement Learning (HRL) can be seen as RL with macro operators, where a macro can call other macros. A macro is a set of primitive operators and in HRL can be seen itself as a primitive operator (like sub-routine calling). From a control point of view a macro is a partial policy, also known as temporally extended action, option, skill, behaviour, mode or activity. Options consist of an input set (set of states) I a policy π and a termination condition β (probability distribution): $< I, \pi, \beta >$. The option is available only if $s \in I$, where s is the current state. If the next state is s' according to $\pi(s, a)$ for some action a, the option terminates with probability $\beta(s')$. Any primitive action can be seen as a one step option (with $\beta(s) = 1, \forall s \in S$). We can have Markov options and semi Markov options. The first depend solely on the current state of the core MDP while the latter on the entire history of states, actions and rewards since the option was initiated. We can define hierarchical options as triples $< I, \mu, \beta >$, where μ is a semi Markov policy over options. An interesting fact (and drawback) is that when an option is invoked we must follow its policy.

The update of Q-Learning [141], one of the standard RL algorithms, regarding options, does nothing until the option terminates and moreover it only applies to one option at a time. This means it is not applicable to non-terminating options. This is the motivation behind intra-option learning methods, such as the one-step intra-option Q-learning. Techniques have also been developed to allow interruption of an option in favour of another more highly valued, or adjusting an option's terminating condition to allow the longest expected execution without sacrificing performance. In most cases, the system designer typically provides prior knowledge, in the form of a specific set of options. Option policies can be seen as policies for efficiently achieving subgoals. Subgoals consist of either one state or a region of the state space and reaching those states facilitates reaching the overall goal. This can be achieved by subgoal values or pseudo-reward functions and one can automatically determine which subgoals are useful, by detecting regions the agent frequently visits on a successful trajectory and infrequently in unsuccessful trajectories (also known as macro-operator discovery).

A Hierarchy of Abstract Machines (HAM) is an approach to structuring MDP policies. It resembles multi layer control of large Markov chains or hybrid control methods. HAMs extend these ideas, by allowing policies to be specified as hierarchies of stochastic abstract finite state machines H_i. Each H_i has four types of states: *action*, *call*, *choice* and *stop*. One may also combine a HAM H and an MDP M, denoted as HoM, which can be seen as a way of restricting the set of policies for M. The only relevant states in determining $HoM's$ optimal policy are choice points and the rest may be discarded. An extension of HAMs are Programmable HAMs (PHAMs) that add interruptions, aborts, local state variables and the ability to pass parameters.

MAXQ [142] is another approach to HRL which does not reduce the entire MDP to a single SMDP but to a hierarchy of SMDPs that can be solved simultaneously. Each SMDP can be seen as a node, representing a task, in a graph. Non-leaf nodes represent subtasks and leaf nodes represent primitive actions. The graph only restricts the action choices whose suitability (whether the task should be executed) or execution order is defined by the parent node's policy. Each subtask has a subtask policy π_i, a set of termination states (the rest are called active states) and a pseudo-reward function that assigns rewards to termination states. Subtasks are very like the aforementioned options, with the addition of pseudo-reward functions. MAXQ explicitly adds a component to each state that is able to return the current contents of a stack that contains parameter names and values of the hierarchy of calling sutbasks, much like programming languages. Consequently, a policy must assign actions to every combination of state and stack contents. A hierarchical policy is defined as $\pi = \{\pi_0, ..., \pi_n\}$, where π_i is the policy of M_i. The hierarchical value function gives the expected return for each state-stack pair. The advantage of pseudo-reward functions is that they enable the system designer to define subgoals that must be achieved, without defining a policy to achieve them. AXQ may also be generalised to to multi options, for problems involving multiple agents [142]. All these approaches, however, assume that the states are fully observable, that the agent acts alone in the environment and that policies are restricted to sequential activities.

3.1.2 Modeling the Dialogue Problem

Various models have been proposed in the literature, that try to effectively capture all aspects of managing the interaction in ADS. In this Section, we review the most prominent ones, which are Finite State Automaton (FSA), Slot Filling / Information Seeking (SF/IS) [85] and Information State Update (ISU) [50]. The FSA is essentially the simplest state machine regarding state representation, since the FSA only stores information about the current state's identifier. Each state may be linked to other states, forming a directed graph, where edges are labeled and cannot change during execution. Transiting to a new state might trigger a system action, such as an utterance or movement of some motors. The benefit of this model is that it is easy to use and understand, but the drawback is that it lacks versatility and cannot be maintained when applied to non-trivial tasks. According to the ISU model, the DM retains a single state, which is updated according to a set of rules, the application order of which is dictated by a control strategy. These rules are triggered by events, such as the user providing input or confirming something. There are no state transitions in this model, which alleviates the state space problem, but on the other hand makes it difficult to predict the effects of rule application, when the rule set grows large. Plan Based systems have a set of goals the DM needs to achieve, and a planner responsible for making the appropriate decisions in order to achieve them. Goals can be ordered as a hierarchy, and the system would need to satisfy all sub-goals of a given goal before considering it satisfied. A benefit of this model is that it allows the system to separate domain dependent from domain independent parts of the discourse, thus allowing the domain independent parts of a DM to be re-used in other applications. The SF/IF model the dialogue problem as a form which contains information that the user needs to fill. Systems adhering to SF/IS, therefore, prompt the user for pieces of information and when enough information is gathered, the system performs the task it was designed for (e.g. hotel booking). Two very interesting variations of this model have been proposed, targeted at addressing the state space problem. Bayesian Update of Dialogue State (BUDS) [90], which is based on Loopy Belief Propagation [91] and the Hidden Information State (HIS) [51] which further develops this idea.

In order to apply machine learning and other techniques it is necessary first to formally define the dialogue problem. A nice point of view is to see it as an optimization problem and model using MDP, as in [45]. The state space in their formulation corresponds to the knowledge the system has about its environment. The action space consists of the actions the system is able to perform, for example interact with the user or search a database. A dialogue session (or episode in other works) is a path through the state space, from the initial state to a terminal state (goal or not). Last, a dialogue policy defines the next action to take, given the current state of the system. In the next section we briefly define the dialogue problem, using the Information Seeking paradigm, as this is what we will follow in the rest of this work.

3.1.3 Information Seeking Paradigm

Many researchers, when using RL to train their DS, model the dialogue problem using the *slot filling problem* paradigm. Before delving into the details, we present an example to showcase the main idea. Consider a flight booking ADS, where users need to provide information such as dates of travel, price range, destination etc. The user's goal in this case could be summarized as: "I need to book a flight from *Chicago* to *New York*, in *business* class, on *August 17th* and return on *September 1st*". Out of this sentence the system only needs the underlined words in order to perform the booking. We can generalise this user's goal and create a template, such as: "I need to book a flight from [____] to [____], in [____] class, on [____] and return on [____]". These blanks are therefore called *slots* and the user needs to provide a (usually discrete) value for each one, i.e. fill the slots. The system on the other hand needs to provide incentive to the user, by asking questions and requesting slot values, for example "When do you plan to return?".

A slot filling problem has a set of slots $Z = \{z_1, ..., z_N\}$, where each slot z_i can take values from $M_i, i = 1, ..., N$ and a set of (system) actions $A \in \{1, ..., N\}$ that are used to represent requests for slot values, greetings, database retrieval, etc. In this formulation we let a_i request the value of slot z_i, where if the action is "database retrieval", the corresponding slot would mean that the item has been retrieved from the database. For our experiments we always let the last action a_N to mean "present the results to the user", which terminates the episode. The state of the dialogue system is represented by a dialogue state $d \in D$, where D is the set of all possible dialogue states. The dialogue state is a vector of dimension N, where each dimension i holds the value of slot z_i: $d =< z_1, ..., z_N >\in M_1 \times ... \times M_N$. We can also define which actions will be available in each dialogue state. To this end we define a subset of actions $\tilde{a_i} \subset A$ to be the set of available actions for dialogue state d_i. An Availability Matrix, that shows which actions are available in every dialogue state, is defined as:

Definition 1 (Availability Matrix) *The Availability Matrix* $\tilde{A} \in \{0, 1\}^{|D| \times |A|}$ *represents available actions at each dialogue state and is defined as:*

$$\tilde{A}_{ij} = \begin{cases} 1, & \textit{if} \quad a_j \in \tilde{a}_i \\ 0, & \textit{if} \quad a_j \notin \tilde{a}_i \end{cases}$$

where $a_j \in A$ *and* $\tilde{a}_i \subset A$.

In order to have a rough measure of how many actions are available at each dialogue state and therefore a measure of how difficult the problem is, for our experiments, we define the following metric:

Definition 2 (Density) *Density is defined as the ratio of the number of available actions to the total number of actions:*

$$\frac{|\{(i, j)/\tilde{A}_{ij} = 1\}|}{|D| \times |A|}$$

As we mentioned before, in this work we are actually solving the information seeking problem. In such a setting there typically is an optimal set of slots whose values are necessary in order to meet the user's goals. For example the set of slots could represent fields in a database that are necessary to retrieve an item with good accuracy (not too many items or a wrong item). We therefore define this optimal subset as the user's query $q \subset Z$, which is hidden from the system. It thus needs to learn to exactly ask for the value of the slots in q.

If, while designing $\tilde{A}$, we end up with a slot filling problem where one or more dialogue states have no available actions, or one or more actions are not available in any dialogue state, we would practically have a disconnected MDP and a problem that cannot be solved (at least not for all possible q). An availability matrix $\tilde{A}$ is called *admissible* if it the MDP derived from it is not disconnected and it is called *inadmissible* in the opposite case. To make sure we do not create an unsolvable slot filling problem we should let each dialogue state have at least one available action and let each action be available from at least one dialogue state, as stated by the following rules (or guidelines):

$$\exists \tilde{A}_{ij} = 1, \quad 1 \leq i < |D|, \forall j \tag{3.4}$$

$$\exists \tilde{A}_{ij} = 1, \quad 1 < j < |A|, \forall i \tag{3.5}$$

where $j > 1$ since we begin from d_1. In our modelling we also allow *Show Results* to be available from any state:

$$\tilde{A}_{i|A|} = 1, \quad 1 \leq i \leq |D| \tag{3.6}$$

To model errors, such as misunderstandings, we let the system transit from state to state according to a probability distribution P_t, that is defined as:

$$P_t(d_k|d_i, a_m) = \begin{cases} p_j, & k = j \\ \frac{1-p_j}{|D|-1}, & k \neq j \end{cases} \tag{3.7}$$

Assuming that with no noise and errors, the system would move from state d_i to state d_j by taking action a_m. To model noise and errors the designer defines the probability of a correct transition, p_j and the remaining probability mass is uniformly distributed across the rest of the dialogue states. In our model we also add a small noise ν to $P_t(d_k|d_i, a_m)$ at the end of each episode, to take into account unexpected events.

In the present work we have extended this model with *confirmation* actions, meaning that for each slot the system can either ask for its value or confirm its value. We therefore have $2|Z|-1$ actions in total, and $z_i \in \{0, 1, 2\}$ to model the status of the slot as empty, filled with high confidence and filled with low confidence, respectively. This simple extension, for a system with 7 slots, leads to $(3^{14-1}-1)^{3^7} \simeq 1.08e{+}13,565$ possible policies for a system with the simplest form of complex actions (composed only of basic actions), versus $(2^7-1)^{2^7} \simeq 1.9e+269$ for the same system without the confirmation actions.

We also replace the confidence model presented in [Papangelis et al. 2012d] and use the following updates:

$$C(z_i) \leftarrow \begin{cases} 1, & \text{initialization} \\ 1 - e_c, & a_i \text{ is `askSlot'} \\ 1 - (1 - C(z_i))^2, & a_i \text{ is `confirmSlot'} \end{cases} \tag{3.8}$$

Where $e_c \in [0, 1]$ is the margin by which we are not confident that slot z_i has been filled correctly (recognition error). $C(z_i)$ represents our confidence that

z_i has been filled correctly. It should be noted that $C(z_i)$ is initialized as 1, as we are confident that in the beginning all slots are empty. a_i is the action that was taken and led to slot z_i being filled or confirmed. The optimal reward that can be achieved then is $-|q|$, assuming we have no recognition errors (so that the system will only use *askSlot* actions). When we have understanding errors (i.e. $e_c > 0$) the optimal reward depends on the probability that the understanding error for each slot is lower or higher than the threshold of confidence set by the designer (above which a slot is considered filled correctly). Please note that when we have complex actions that are composed of more than one confirmation actions we assume that the user correctly confirms / rejects each slot separately (or that the system has a mechanism of inferring it). Otherwise, if the system, for example, requires confirmation for two slots simultaneously and the user simply replies “No”, the system would have no way of knowing to which slot the user was referring to. In this case the system could either reject both slot values or require more information.

3.2 Evaluation of Reinforcement Learning Methods

In our ADS, we apply a Hierarchical Incremental Actor Critic (HIAC) [143] algorithm for dialogue policy learning, which effectively is a Natural Actor Critic (NAC) [92]. Before concluding, however, to use NAC for our system, we conducted extensive evaluations of the most frequently used RL techniques in DM literature, which include basic RL algorithms, such as SARSA and state of the art algorithms, such as NAC. In the following section we present our efforts to gain intuition on the strengths and weaknesses of RL algorithms applied to online dialogue policy learning.

For our evaluation, we designed and used the following reward function:

$$R(d, a_i) = \begin{cases} -1, \quad \text{if} \quad a_i \neq a_N \\ -10|\{d_i | d_i = 2\}| - 100, \quad \text{if} \quad a_i = a_N, \exists j | q_j = \emptyset \\ -10|\{d_i | d_i = 2\}|, \quad \text{if} \quad a_i = a_N, \neg \exists j | q_j = \emptyset \end{cases} \tag{3.9}$$

This reward function assigns a penalty of -1, for every system action except for presenting results, in order to penalize long dialogues. It assigns a penalty of -10 for each slot that has low confidence plus a penalty of -100 if the system attempts to present results without having filled all the necessary slots in q. Last, it assigns a penalty of -10 for each slot that has low confidence at the end of the episode. One could also divide the rewards by $C(z_i) \in [0, 1]$ to provide incentive to the system to confirm slot values that currently have low confidence.

When the system requests the value of a slot, it is filled with high confidence with some probability, according to the user model or the system designer, to model misinterpretations. When a slot is filled with low confidence, the system must learn to confirm its value. Once a confirmation action is taken on such a slot, its confidence value is re-estimated according to (9), and if it is above a threshold, it is considered filled with high confidence. One way to remove this manual threshold is to incorporate $C(z_i)$ into the dialogue state, so that the algorithms will learn to select confirmation actions from states where $C(z_i)$ values are low, and learn to ask for new slots from states where $C(z_i)$ values are high.

The algorithms we chose to implement and evaluate on the SF/IS problem, along with some of their characteristics, are the presented in Table 3.1, where LS-SARSA(λ) stands for Least Squares SARSA(λ), i.e. a version of SARSA(λ) with function approximation and IAC stands for Incremental Actor Critic.

Table 3.1. Online RL algorithms we selected for evaluation in the dialogue policy learning problem.

Algorithm	Model	Policy	Iteration
SARSA(λ)	No	On	Value
LS-SARSA(λ)	No	On	Policy
Q Learning	No	Off	Value
Q(λ)	No	Off	Value
Actor Critic - QV	No	On	Policy
IAC	No	On	Policy
NAC	No	On	Policy
DynaSARSA(λ)	Yes	On	Value
DynaQ	Yes	Off	Value
DynaQ(λ)	Yes	Off	Value
DynaAC-QV	Yes	On	Policy

All the above algorithms are online and can be categorized into model-free and model-based. Model-based algorithms use a model that simulates the world and alternate real and simulated interactions during the learning process. Model-free algorithms, as the name suggests, rely on real interactions only. To implement model-based algorithms we applied the Dyna framework, described in [144]. RL algorithms can also be categorised based on the way they learn good policies. On-policy algorithms follow the policy they are trying to learn, while off-policy algorithms follow one policy while learning about another. RL algorithms can also be characterized as value-iteration or policy-iteration, depending on the way train and evaluate their policy. It should be noted here that IAC is an Actor-Critic algorithm following the gradient of expected rewards, while NAC follows the natural gradient. In the following sections we present our experimental setup and results. This setup will also be used for future evaluations.

3.2.1 Experimental Setup

For our evaluations, we used the information seeking paradigm. We used a noise free setting, a noisy setting and one where there was a change in the environment (modelled by a change in the reward function) and the system needed to adapt again. To simulate interactions with real users we developed a user simulator, which we describe in the following section.

3.2.2 User Simulator

For this series of experiments we developed a user simulator able to interact with a slot filling or information seeking ADS. The goal of the simulator is to provide values to specific slots, that describe its intent. These slots can be varied by the system to model interaction with multiple users. Specific slot values are irrelevant at this point we only care to simulate the fact that users provide information or not. This set of slots is called the *user's query* q, and is hidden from the learning algorithm. The algorithm, thus, must learn which slot values to ask for, in order to get the necessary information (contained in q), and perform the task that meets the (simulated) user's needs. Later versions of this simulator include ASR errors, slot value confidence and other features, that will be discussed in the next Chapter.

3.2.3 Results

Results of our experiments are shown in Figure 3.2, from which we conclude that, for this problem, model-free algorithms perform better than model-based algorithms following the Dyna framework. In terms of absolute time, ActorCritic-QV proved to be significantly faster than the rest, a trait that can prove useful in various cases, such as when operating with limited resources. IAC and NAC were by far the best performing algorithms, although the most computationally expensive ones.

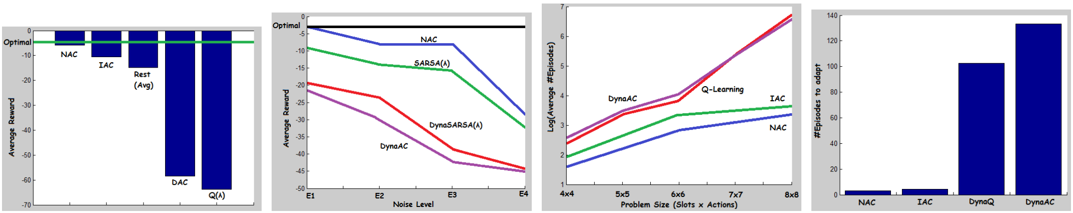


Figure 3.2. Results of our evaluation. In all figures we show the two best performing and two worst performing algorithms..

NAC was consistently better than any other algorithm and our final verdict is that if resources are unlimited, NAC should be the algorithm of choice.

This work resulted in a publication in the 8th Language Resources and Evaluation Conference (LREC) [145] and a publication in the 13th European Chapter for the Association of Computational Linguistics (EACL) [146]. Both works presented in this section resulted in the development of a SF/IS ADS, based on the Olympus [80] platform, which was published in the 7th Hellenic Conference on Artificial Intelligence (SETN) [2] and acts as a guide in a virtual museum. The system interacts with the users and prompts them for exhibit features they are interested in, such as era, geographical location, etc., and when it has enough information (i.e. when the number of hits in the database is below a certain small number) it presents matching exhibit descriptions to the users. The system was implemented using the Olympus [80] platform, which is an open source DS development framework. To learn good dialogue policies, we implemented various learning algorithms, which are interchangeable to accommodate changing needs of the system, for example amount of available resources.

3.3 Discussion

In this chapter, we presented necessary background knowledge, as well as how we model the interaction. We also presented an extensive evaluation of online RL techniques (model free or model based) and described in detail our findings and intuition.

In the following chapter, we present our contribution to the state of the art in online dialogue management. More specifically, we present two novel algorithms for combining solutions to simple tasks into solutions to more complicated ones and for estimating the user's reaction to system actions in order to be able to guide the conversation and achieve the system's goals.

CHAPTER 4

Towards Complex Behaving Adaptive Dialogue Systems

In this chapter we present our approach on achieving complex behaviour in ADS and our contributions to the field of Adaptive Dialogue Management [147]. We describe in detail our proposed ADS, which employs HRL techniques for learning and adaptation. More specifically, we propose two novel online methods, able to combine system actions and solve complicated tasks and able to estimate the effects of those actions on the user and the environment (or the dialogue state in general). Our system is able to learn a dialogue policy that achieves its mandatory goals and also achieves as many of the optional goals as possible, and is also able to learn how to combine basic or complex system actions into more complex ones, in order to solve very complicated tasks.

4.1 Motivation

Dialogue Systems (DS) have attracted the research community's and the industry's interest, during the last years, as they exhibit a number of benefits, such as being cost effective, providing an easy way to collect statistical data and many others. A typical DS is used for providing a service, such as customer support or flight booking. Such a system can greatly cut costs for industry and waiting times for customers. Trying to make DS more natural and human-like, researchers have developed techniques that enable them to adapt to specific users or user groups, circumstances, environmental changes etc. As shown in Figure 1, a DS is composed of modules such as Automatic Speech Recognition (ASR), Natural Language Un-

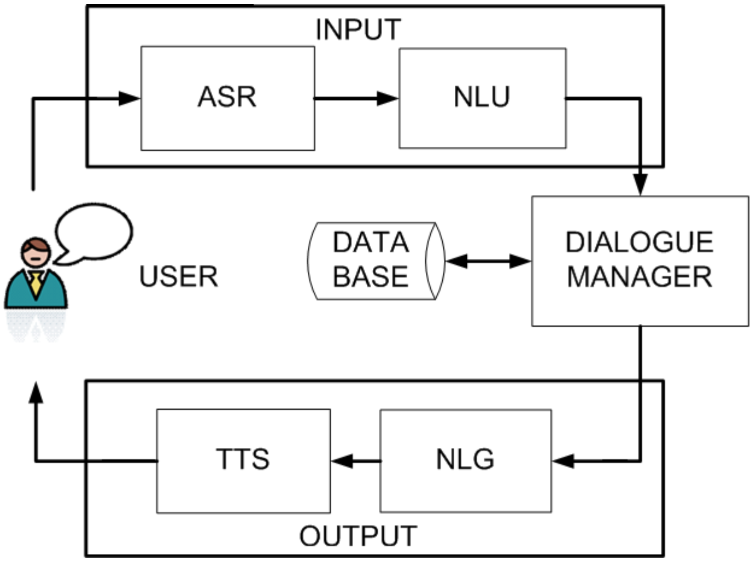


Figure 4.1. A Dialogue System sample architecture..

derstanding (NLU), Dialogue Management (DM), Referring Expression Generation (REG), Natural Language Generation (NLG), Text To Speech (TTS) and others (not shown). During the last two decades DS have matured significantly, from simply responding with canned text, to intelligently processing the user's input and responding appropriately. Industry has also begun to trust these systems more and we can see many commercially available DS being offered and many that are already being applied in a variety of fields, such as customer support or providing information.

These systems, however, are somewhat rigid and inflexible to a great extend, meaning that once the design phase is finished, not much can be done to alter their behaviour without disturbing the service. Even most of the systems that use Machine Learning (ML) techniques cannot easily adapt to changes in their environment, as they are trained before being put to use and follow static behaviours from that point on. Moreover, the interaction is not very natural or human-like, as the system follows a predefined dialogue policy and cannot adapt to each individual user or to unexpected events, that are not described by the system's dialogue policy, which is responsible for realising the system's behaviour. Trying to overcome the limitations of DS, researchers have been very actively working on achieving adaptation in several components of a DS, such as ASR, NLG or DM. Adaptive Dialogue Systems (ADS) are therefore DS which are able to adapt to environmental conditions, adapt to specific users and to their changing needs. The main focus of this paper is on achieving adaptation in the DM, by combining online learning algorithms (i.e. which perform their updates after each dialogue move), such as SARSA or Actor-Critic, with batch learning algorithms (i.e. which perform their updates after each episode). More specifically we present two batch learning algorithms, one that is able to learn how to combine basic and complex system actions into more complex ones, allowing it to solve complicated problems. For example, in an assistive robot, in order to serve the user, the system can combine the basic actions "greetUser", "navigateToFridge", "graspWater", "navigateToUser" into "fetchWater" that is composed of all of them. Another benefit of this approach is that the algorithm can learn different ways to achieve the same task, depending on the feedback it receives (e.g., user preferences). The second technique we present, is able to estimate the effects of those actions on the user and the environment, therefore enabling the system to guide the learning algorithm towards achieving its goals. Both techniques, thus, help the system exhibit more complicated and intelligent behaviour, taking a step towards better user experience.

4.2 Related Work

Complex actions in this setting are combinations of basic or other complex actions, i.e. sub dialogues or tasks. A complex action thus could be represented as a tree, where basic actions are at the leaf nodes and each parent node is a complex action composed of its children. For example, in the system proposed by [49], a travel planning DS, the user can book flights, hotels and rent cars. A basic system action in such a system would be asking for departure and arrival dates, pricing range, rental car specifications, etc. The user, on the other hand, can perform actions such as provide flight departure and arrival dates, ask about the availability of a specific car for specific dates, or provide information that the system has requested. A complex action then would combine basic and other complex actions to solve more complicated tasks, such as travel planning. Flight booking, hotel booking and renting a car would then be complex actions, which could be combined to solve the travel planning task. This ability to combine actions into more complex ones is a huge benefit, as it allows the system to reuse previous knowledge. For example, a complex action that requests dates could be re-used in flight booking, hotel booking and car rental. Using complex action learning, the system can learn which is the best way to plan a journey for specific groups of people, such as tourists, students, sports fans or business men. A businessman, for example, would most likely want to rent a car, while a student would probably not. Being online, the system can also adapt to each user's specific needs and goals. More formally, the goal of complex action learning is to find the optimal set of complex actions that will lead to the highest possible user satisfaction. Another example is an ADS that controls a robot's actuators [148]. In this setting, basic actions can be *Verbal*, *Computational* or *Physical*, corresponding to the robot saying something, deliberating about something (e.g., plan an arm motion) or moving a motor to control an actuator. Assuming the robot can manipulate objects, a complex action could be "buildTower", which would be composed of a *Verbal* action "askColour", many *Physical* "pick" and "place" actions and many *Computational* actions to plan the motions. There have been efforts in the literature to learn complex (or hybrid) actions, such as [114, 62] but to the best of our knowledge only [2] present a complex action learning algorithm, able learn in a batch learning fashion.

Chandramohan and Pietquin [114] propose a Spoken Dialogue System (SDS) with complex actions that provides information about restaurants. The authors predefine which complex actions will be available to the system and conclude that their system performs better with complex actions compared to using only basic ones. They advise using basic actions in the presence of noise in ASR and also state that complex actions are more suitable for experienced users. Our approach is different in that our system is able to learn how to combine basic actions into complex actions (or combine basic and complex actions into "larger" complex actions etc).

Cuayahuitl et al. [49], model the dialogue problem using hierarchical structures and apply hierarchical reinforcement learning (HRL) techniques to solve it. They are able to represent the dialogue state at various levels of detail and actions can be either basic or complex. The authors model complex actions using Semi Markov Decision Processes (SMDP) and use HRL to find an optimal dialogue policy. They also use Hierarchical Abstract Machines which control deterministic transitions of the system and may therefore model prior knowledge. In another work of Cuayahuitl et al. [62], the authors use multi-agent HRL methods, aiming to align verbal and non-verbal actions in a robot DS. They use a hierarchical statistical model to represent the interaction, that uses joint verbal and non-verbal actions. The difference with our approach is that we learn how to combine actions, with no restriction of simultaneous execution of one *Verbal* and one *Physical* action. Lemon [5] proposes to use HRL to find optimal dialogue policies and optimise NLG output simultaneously. He tackles the state space scalability problem with function approximation, which also allows policies to be generalised to new states.

Achieving the system's goals has been traditionally handled by the research community by either incorporating the goals in the environmental feedback procedure or by designing the DS according to the goals. Another method proposed in the past by [2] is User State Estimation (USE), that is able to predict the effects of system actions on the environment and, therefore, make a more informed decision on the best way to react to the user's input, in order to achieve the system's goals. Using these methods, the system is able to make an estimate on the user's short-term and long-term reaction to its actions. In this paper we present this method in detail, as well as results from extensive evaluations.

In the following Section, we provide some necessary background knowledge and in Section 3 we describe how we modelled the dialogue management problem. In Section 4 we present our complex action learning algorithm and in Section 5 we present our action effect estimation algorithm. In Section 6 we describe the experimental setup and the results of our evaluation, and in Section 7 we present the implementation of an ADS where our techniques are applied. Last, in Section 8 we conclude this paper and briefly discuss our future plans.

4.2.1 Hierarchical Reinforcement Learning

A way to better understand how HRL works is to see it as RL with macro actions. A macro action can call simple actions or other macro actions. Macro actions are also called options, and consist of an input set of states I, a termination condition β, which is a probability distribution and a policy π. For the rest of the paper we will refer to macro actions as complex actions, to be consistent with DS terminology. If the current state s_t belongs to the input set I_k of a complex action k, then that complex action is available from s_t. Once a complex action has been evoked, it receives focus and starts executing its actions, according to its policy π_k. At each time step t', the complex action will terminate with probability $\beta(s_{t'})$. A simple action, then, can be seen as a complex action with $\beta(s) = 1, \forall s \in I$. Note here that once the system selects a complex action, it must follow its policy until it terminates. Techniques, however, have been developed to allow interruption of a complex action in favour of another, more highly valued, or to allow adjusting a complex action's terminating condition. [142]

The system designer, in such a setting, typically provides prior knowledge in the form of a specific set of complex actions (semi-learnt policies, etc.). We propose an algorithm that is able to rank all possible complex actions and select the top-k performing ones, without the need to explicitly provide prior knowledge. This technique will be better described in Section 4.

4.2.2 Complex Actions

In order to gain intuition on how complex actions and complex action learning can be applied to DS, consider the object manipulation ADS, that we mentioned in Section 1, and suppose the environment consists of a table and several blocks that can be stacked on top of each other. The *Build* complex action is composed of the following actions:

- Ask Shape
- Ask Colour
- Stack { Pick { Plan, Grasp }, Place { Plan, Put } }

This would be represented in our system as sets of basic and complex actions, for example:

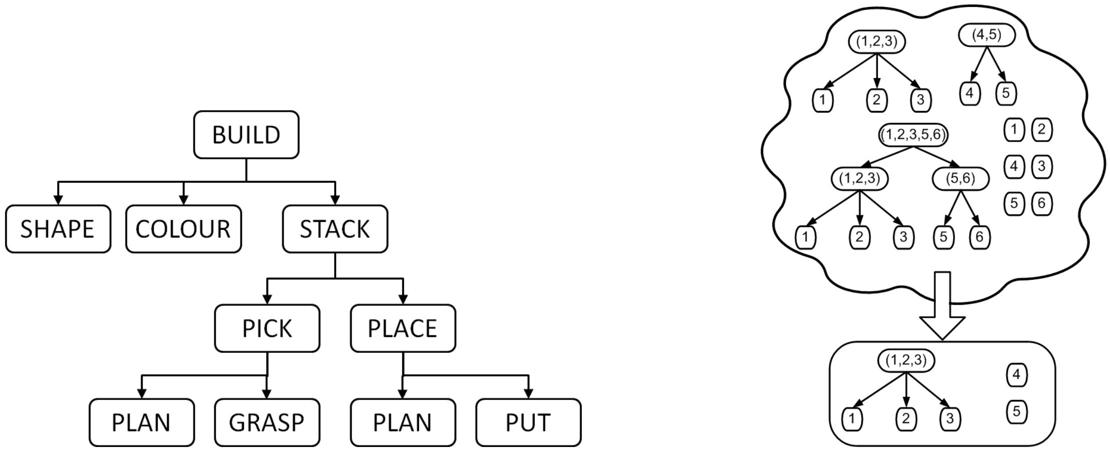
$$a_0 = \{a_1, a_2, \{\{a_3, a_4\}, \{a_5, a_6\}\}\}$$

where action a_0 is composed of two basic actions, a_1 and a_2, and one complex action, that is in turn composed of two complex actions, who are composed of two basic actions each. It should be noted here, that there is no limit on the ways to combine complex and basic actions, and we can also re-use the same basic action in several complex ones that are within a bigger complex action. This representation of *Build* could be visually interpreted as a tree, as shown in Figure 4.2(a). It is important to stress again that complex actions are re-usable. The *Pick* complex action, for instance, could be re-used for selecting objects of the same colour or sorting objects according to their colour, etc. One can imagine that this benefit is even greater when many complex actions are being re-used to create complicated system behaviours.

To summarize, we model the dialogue problem with complex actions, we used a hierarchical MDP where P_t corresponds to transition probabilities, the dialogue states D correspond to the MDP's states S and the system's actions (requests for slot values, greetings, etc), basic or complex, correspond to the MDP's actions A. Having defined the *slot filling problem* and complex actions, we can now see how we can learn optimal dialogue policies.

4.3 Complex action learning

In this section, we describe a complex action learning algorithm, called Action Weights Learning (AWL) which was first proposed in [2] and is here extend and evaluated on a harder problem, as presented in Section 3.1. In a real world ADS, that we wish to be as human-like as possible, learning complex actions is an important part of dialogue policy learning. Without complex actions, the system would appear simplistic and not very intelligent. Manually creating dialogue policies with complex actions, without a learning mechanism, is as well unfeasible, as the amount of effort required to create and maintain such hand-crafted dialogue policies is prohibitive. The



(a) A complex action

(b) Complex action learning

Figure 4.2. A sample complex action (a) and the procedure of complex action learning (b).

presented algorithm is, to the best of our knowledge, the first online batch complex action learning algorithm applied in ADS. Being online has the benefit that the system does not follow a static policy when interacting with users, but is able to adapt to events that occur in its environment, adapt to specific users as well as their current needs. AWL uses a ranking scheme to evaluate complex actions that are used by the system and assess their contribution to the total reward achieved for a specific episode. Each complex action, then, has a score (or weight) that shows how likely it is to achieve good rewards if the system uses it. It should be noted here that each action is evaluated at the end of each episode, as an action may incur bad immediate rewards but lead to a state from which the optimal reward can be achieved. This algorithm does not make decisions on which action the system should perform next, it only assesses the performance of each action and therefore can be used along with any HRL algorithm.

Having assessed each action's performance, in a new episode we can select the top-k performing actions and let the system choose the next action from that set only.

The designer, however, also has the option of allowing the selection of actions outside the top-k set, with a small probability, so as not to overfit (that action will have the chance to increase its performance score and may thus be selected in the top-k in a later episode). This algorithm can thus tackle the problem of dealing with very large action spaces, when allowing complex actions. If we only consider the simplest form of complex actions, which means complex actions composed of basic actions only (and not other complex actions - one can view them as a trees of height 2), the possible complex actions derived from 8 basic ones is $2^8 - 1 = 255$, excluding the "empty" action. One can imagine that for more complex actions or when we have more basic actions available, the number of possible complex actions grows rapidly. The policy space also grows very fast; consider that for a DS with 8 binary slots and basic actions only, we have $2^8 = 256$ dialogue states and $7^{2^7} \simeq 1.5 \cdot 10^{108}$ possible policies, versus $(2^7 - 1)^{2^7} \simeq 1.9 \cdot 10^{269}$ possible policies when we allow the simplest form of complex actions.

In the present work we extended AWL in order to automatically compute k after each episode, i.e. to estimate how many actions will be necessary in the next episode. The way to achieve this is to sort the action weights in a descending order and then try to find a cluster that contains the highest weights. This is done by comparing the difference of two consecutive weights with the average (so far) difference between weights of actions in the top-k set. The initial value for k may be equal to $|A|$ or any other value the designer finds appropriate. Algorithm 5 describes AWL, where $A_T \in \{0,1\}^{|A|}$ are the action traces, meaning that $A_T(j) = 1$ for every action a_j that was used in the ith episode and $A_T(k) = 0$ for every action a_k that was not used. The sum of A_T is in the S_{A_T} matrix, where $S_{A_T}(j)$ gives us the number of times action a_j has been used so far. The parameter k is used to select the top-k performing actions. The weights are updated according to the equation in line 8. This automatic update procedure incurs some computational cost, so the resulting complexity of AWL is increased a little but still remains linear in the number of actions $|A|$.

It is important to note here that setting a very low value for k may lead to an inadmissible action set. This is because the system may need the value of certain slots for which there are no actions, because only the top-k were available, and k was too low. If the designer does not wish to automatically tune k, from our experience it is good practice to begin with a relatively high k (for example $|A|/2$) and decay it as the episodes progress, down to a certain limit (e.g. until $k = |Z|$). AWL seems to perform well this way, although it exhibits highly negative rewards in the first few episodes, as too many complex actions are available and must be used in order to be assessed and complex actions generally have greater impact on the accumulated rewards than basic actions. Last, it is also important to note that AWL is sensitive to initialization of w, as if it is initialized randomly, for example, the algorithm will need many more iterations, where random actions will be selected for the action set, before it can converge to the optimal w and therefore to the optimal action set.

4.4 Goal Achievement

In this section we will describe a technique, that has also been proposed in [2] and in this work is extensively evaluated in a more realistic and challenging scenario, which allows us to estimate the effects of system actions on the user and the environment, called User State Estimation (USE). This technique essentially provides us with an estimate of how the user might react if we select a certain action. We can thus make a better decision, depending on which goals we want the system to achieve (e.g. cheer the user up, teach the user something new etc.) or even adapt to its environment or to user's preferences and so on. This method can also prove very useful in forming long-term relationships between the system and its users, by knowing how to react to certain user behaviours.

The main idea behind this algorithm is that we can predict what the next state of the environment will be, depending on which action we take and create a probability distribution of the next environmental state u_{t+1}, given the last system action a_t and the current dialogue state d_t: $\tilde{P}(u_{t+1} = g|d_t, a_t)$. The state of the environment (or the next input to the system) is defined as a discrete vector: $u \in U = U_1 \times U_2 \times ... \times U_L$, where each dimension $u^{(i)}$ can take values from U_i and can model anything of interest, for example NLU output, the user's emotional state or facial expression, his/her preferences, information about the environment that is not directly related to the user and so on. g represents a desired state that we want the environment to be in, for example a desired emotional state (happy), a desired facial expression (smiling) and so on. Having calculated $\tilde{P}$, we can use the estimates to choose those actions that have the highest probability to achieve the desired state (or goal) g that we are interested in. We can think of this as guiding the algorithm through the policy space, in order to find a solution that would achieve our goals, i.e. a desirable instantiation of u, that would represent that the system's goals have been fulfilled.

As with AWL, USE also works in conjunction with a HRL algorithm. In our experiments we used a Hierarchical Incremental Actor Critic (HIAC) algorithm, as described in [143]. This algorithm maximizes the cumulative expected rewards J and USE extends it to achieve g as well, when possible. This is done by selecting an optimal system action, at each turn, which has the highest probability, according to $\tilde{P}$ to achieve g. USE then will maximise J and thus meet the goals set by the reward function $R(d, a)$ and will also try to find a dialogue policy that will eventually lead the environmental state to g. If we think of $R(d, a)$ as modelling hard constraints and g as modelling soft constraints, we can think of USE as trying to satisfy the hard constraints and at the same time as having a bias towards satisfying the soft constraints. USE will therefore find an optimal dialogue policy w.r.t $R(d, a)$ that also satisfies the goal g, i.e. it will look for a policy that satisfies g within a set of optimal dialogue policies. Figure 4.3 depicts this schematically, where Goal 1 and Goal 2 represent dialogue policies that lead to desirable environmental states and the S_1 and S_2 represent the policies that the system converged to. For Goal 1, the system is able to find an optimal dialogue policy w.r.t $R(d, a)$ that also achieves g while for Goal 2 no such policy exists and the system converges to the optimal dialogue policy that is closest to also achieving Goal 2. USE will keep looking for such an optimal policy and if it appears it will converge to it. An algorithm that optimizes the dialogue policy together with model parameters (various parameters of the ADS) is the Natural Actor-Belief Critic [Peters et al. 2005]. USE, however, does not optimize parameters, but estimates what the user's state will be next, given that the system is in a certain state and chooses a certain action and USE is also able to guide the system to an optimal dialogue policy that will also achieve a desirable environmental state.

We can now describe USE more formally and present some of the details. As mentioned before, USE samples a probability distribution $\tilde{P}(u_{t+1}|d_t, a_t)$, which gives us the probability of the next state of the environment, given the current dialogue state and the last system action. In effect, USE observes the next environmental state u_{t+1}, while knowing the current dialogue state d_t and the last system action a_t. It therefore updates its estimate for $\tilde{P}$, after each system action. The distribution $\tilde{P}$ is initialized as: $\tilde{P}(u_1^{(i)}|d_0, a_0) \leftarrow \frac{1}{|V_i|}, \forall i$. USE also assumes that the values of the environment state vector are normalized so that $u^{(i)} \in [0, 1], \forall i$. The required modification of HIAC, for example, in order to incorporate USE, is at the action selection

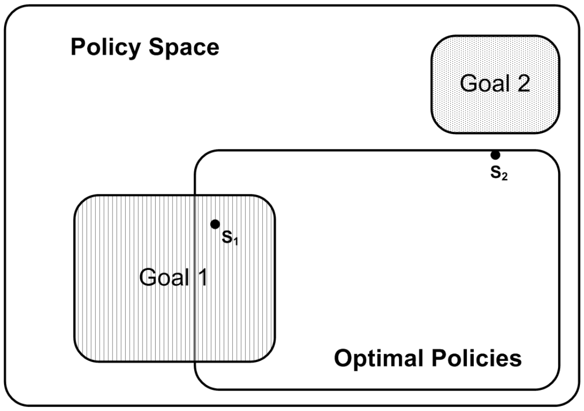


Figure 4.3. USE will attempt to find a policy that satisfies both hard and soft constraints, as in the case when the soft constraints are satisfied by the policies enclosed by Goal 1, where the algorithm would converge to a policy such as S_1. When no such policy exists, as in the case where the soft constraints are satisfied by Goal 2, the algorithm will find a policy such as S_2 that satisfies the hard constraints and is as close as possible to also satisfying the soft ones.

step, and is the following:

$$a_{t+1} \leftarrow max_{\alpha}\{\pi(\alpha|d_{t+1})\tilde{P}(u_{t+1}^{(i)} = g|d_t, \alpha)\} \tag{4.1}$$

With this formulation we can express complex goals, for example:

$$\tilde{P}(u_{t+1}^{(1)} = tense \cap u_{t+1}^{(2)} = low \cap u_{t+1}^{(3)} = neutral|d_t, a_t)$$

or express more complex goals:

$$\tilde{P}(u_{t+1}^{(1)} > 0.19 \cap u_{t+1}^{(2)} \leq 0.45 \cap 0.64 \leq u_{t+1}^{(3)} \leq 0.1|d_t, a_t).$$

It should be noted here that it is up to the designer to assign weights to each sub-goal, to model their significance. USE is also able to handle changing goals, as in that case and assuming g' is the new goal, the algorithm will update $\tilde{P}(u_t^{(i)} = g'|d_t, a_t)$, instead of updating $\tilde{P}$ for the old goal. This however would require several episodes in order to have good enough estimates and guide the learning algorithm towards the new goal. Trying to eliminate this, we can store optimal policies that USE found for previous goals and when a new goal is requested we retrieve the policy that is closest to achieving it. For example, if the new goal is $g = happy$ and no optimal policy exists yet for it, we would retrieve a policy that achieves *elated* versus a policy that achieves *sad*. This way USE does not begin from scratch every time g changes (assuming there is an available policy in storage).

The distribution $\tilde{P}$ is updated according to the following rule:

$$\tilde{P}(u_{t+1}^{(i)}|d_t, a_t) \leftarrow \frac{G(filter(u_{t+1}^{(i)}), \sigma^2) + \tilde{P}(u_{t+1}^{(i)}|d_t, a_t)i}{i+1} \tag{4.2}$$

where $filter(\cdot)$ is a function that takes as input $u_t^{(i)}$ and outputs a discrete value, for example $filter(0.53) = frowning$. Function G takes as input the discretised state of the environment, $u_t^{(i)}$, and a variance value σ^2. It is defined as:

$$G(\mu, \sigma^2) =< N(v_1, \mu, \sigma^2), ..., N(v_k, \mu, \sigma^2) >, v \in U_i \tag{4.3}$$

where U_i are the possible values of $u^{(i)}$ and $N(v, \mu, \sigma^2)$ is the value of the normal distribution with mean μ and variance σ^2 at v. G is used to disseminate the update on $u^{(i)}$ across the rest values of that dimension. This allows us to also update environmental states that are similar (close by) to $u^{(i)}$ and this is useful for a number of reasons. Imagine for example that we want to achieve $g = halfSmile$, and our system also recognises *smiling* and *neutral* (among other facial expressions) which are the closest to $halfSmile$. When the system then finds an optimal policy that yields $u^{(i)} = smiling$ and it updates $\tilde{P}$ for state-actions that lead to that policy, it makes sense to tell the system that $halfSmile$ is very close to *smiling*. The way to do so is by updating $\tilde{P}(u_t^{(i)} = halfSmile|d_t, a_t)$ as well, but with a smaller "weight" as $halfSmile$ is not exactly what we are looking for. This "weight" is controlled by σ^2. The system then will be able to find an optimal policy that yields $u^{(i)} = halfSmile$ much faster. Another benefit is that if no such (optimal) policy exists, the system will converge to the closest one w.r.t. g, much faster. The parameter σ^2 then is a tunable parameter of USE and may be increased or decreased as the episodes progress. For example if no optimal policy that achieves g has been found for some episodes, the system could increase σ^2, or if the goal g changes over time, σ^2 should also be changed. Note here that σ^2 can be hard to tune, especially if we have no similarity or distance function to provide some measure and USE is somewhat sensitive to the value of σ^2. A rule of thumb is that when switching to a very similar goal the designer should set a small σ^2, as the new goal is "nearby" and therefore we do not need much exploration. On the other hand, for goals that are farther away, the designer should set a larger σ^2 value and for goals that might be suboptimal w.r.t $R(s, a)$ the designer should set an even higher σ^2 value so that the algorithm will explore the policy space more and find a policy that is as close to achieving g as possible. In our experiments we decay σ^2 as the episodes progress, as we need the algorithm to gradually focus on a specific part of the policy space, where the optimal policy we are looking for lies. Also, as mentioned before, the system stores previously found policies for each goal g that was set, in order not to begin from scratch when g is requested again. In the same sense, σ^2 values can also be stored for each goal, and are retrieved when that goal is requested again.

Before discussing our experimental setup and the evaluation of AWL and USE, we present the learning architecture of our system. The resulting algorithm, named

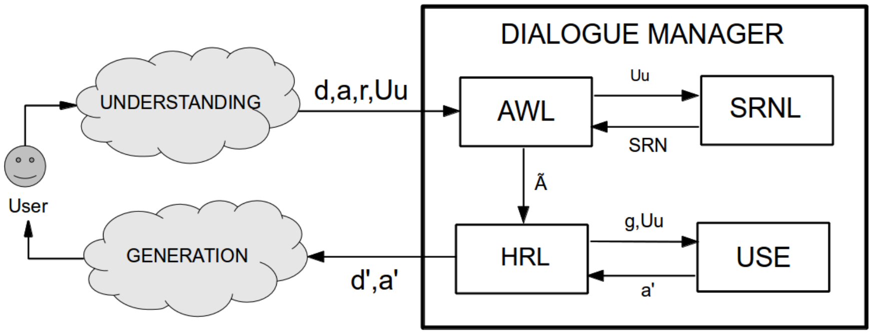


Figure 4.4. The learning architecture of our Adaptive Dialogue System.

AWL-USE is able to rank complex actions, select the top-k performing and use them to guide the algorithm towards an optimal policy that also achieves the goal g.

The architecture is presented in Figure 5.3, where d' is the new dialogue state, a' is the new action, d and a are the old dialogue state and old action respectively, r is the immediate reward received after taking action a from d and u_u is the resulting user state. SRN is the Slot Relativity Network, that contains information about which slots makes sense to be requested together, without confusing the user and will be presented in the following section.

4.5 Evaluation

In order to evaluate the proposed algorithms, we modified the Agenda-based User Simulator [124] and added some tunable parameters (user expertise level, confirmation probabilities and ASR), aiming to avoid overfitting. User simulators have many advantages, such as being fast (RL algorithms require many iterations to converge, especially for high dimensional problems), cost-efficient and many more. We used the same user simulator, with different settings, to evaluate both algorithms.

4.5.1 User simulator

In this section, we describe some modifications we applied to the Agenda-based User Simulator, proposed by [124], that we used to train and evaluate our system. More specifically, this simulator was modified to also keep track of some extra variables, such as user's emotional state, ASR errors and user *attendance level*, for evaluation purposes only. We chose to incorporate ASR errors in the user simulator to model the fact that some users may or may not speak clearly. This was modelled with the recognition error e_c and the confidence model that we presented in Section 2. In general, it is desirable to have as realistic a model as possible, in order for the learnt dialogue policies to apply as well as possible to real users. The simulator interacts with the system at the intention level, bypassing ASR and NLU, NLG and TTS and assuming that the simulator directly passes its intentions to the system. To model ASR errors, therefore, we assign low confidence scores on the slots where an error should have occurred. The modified simulator also has some extra internal variables, that represent *valence* and *arousal* to model the simulated user's emotional state and *expertise* to model the fact that a user may be an expert or a novice. A user state is thus a three dimensional vector: $u =< valence, arousal, expertise >$, where $valence \in [0, 1]$, $arousal \in [0, 1]$ and $expertise = \{Novice, Expert\}$ and can be thought of as an extension to the simulator's dialogue state. The reason we wanted the simulator to have an emotional state was simply to use as a proof of concept, for the experiments of USE, showing that it can select the appropriate dialogue policy that achieves the specified goals. As in [2], we classify (through the $filter$ function) valence and arousal values into 16 emotional states, as proposed by [149]. A simulated user may therefore be in one of the following emotional states: happy, elated, excited, alert, tense, nervous, stressed, upset, sad, depressed, bored, sleepy, calm, relaxed, serene and contented. The simulator's expertise level is used by the system to penalise actions, according to their complexity. Intuition behind this is the fact that an experienced user may prefer complex actions (not overly complex though) in order to finish his/her task faster and easier, while a novice user probably would prefer simpler actions. To model confirmation or rejection of slot values (when requested by the system) we define a static probability, according to which the system will confirm or reject the slot value.

In the present work we modified the user simulator to also model attention levels, to show how USE can simultaneously achieve two independent goals. Again the attention level is just an example of what the system's goals might correspond to. The environment's state thus becomes a four dimensional vector: u = <*valence, arousal, attention, expertise*>, where *attencance* $\in \{payingAtt, notPayingAtt\}$. From now on, for the emotional state which is defined by valence and arousal (i.e. $u_{t+1}^{(1)}$ and $u_{t+1}^{(2)}$), we will use $u_{t+1}^{(1)} =< emotion >$ instead of $(u_{t+1}^{(1)}, u_{t+1}^{(2)}) =< emotion >$, abusing notation, for the sake of brevity and readability. We will also use $u_{t+1}^{(2)}$ for attention. Another aspect added to the simulator, is that a user, whether experienced or novice, would not tolerate complex actions (requests for multiple slots) if the slots requested are irrelevant to each other, for example *"What is your age and preferred drop-off location for your rental car?"*, as the system would seem unnatural. According to each user type then, the simulator has different probabilities of tolerating complex actions or not (for more details see [2]). Valence and arousal values are updated according to the rules presented in Table 4.1, taken from [2] and attention is updated according to a very simple model that yields high attention values for certain emotional states (happy, elated, excited, alert, tense) and low attention values for the remaining emotional states (nervous, stressed, upset, sad, depressed, bored, sleepy, calm, relaxed, serene and contented).

Table 4.1. Update rules of our user simulation model, from [2]

User Model Update Rules		
Update	Valence	Arousal
↑	u_s is relative no system repetitions accurate results	system asks questions
↓	u_s is not relative system repetitions inaccurate results	system is idle too many dialogue turns

As mentioned before, the user simulator has a probability of tolerating complex actions according to the slots requested. In order to define which slots make sense to be requested together by the system, we create a graph (or network) where vertices represent slots and arcs represent their relationship. Weights on the arcs model how significant that relationship is. The way this network, called the Slot Relativity Network (SRN), is created is quite simple. We just mark which slots have been mentioned in the same user utterance and update the weights on the arcs accordingly. More formally, the updates are as follows:

$$w_{i,j} = \frac{|\{z_i, z_j\} \subset q, \forall q \in W|}{|W|} \in [0, 1] \tag{4.4}$$

where the user query q was defined in Section 2, and we let W be the set of all queries made so far. We can now give the following definition, according to [2]: *"If the Slot Relativity Network that is created by the slots $q_i \in q$ of a user query q is connected, then q is called* relative. *Otherwise q is called* non relative.*"*

In order not to overfit the simulator, we varied its parameters before each episode, to model different users interacting with the system. More specifically, we varied the user's expertise level, ASR errors and confirmation probabilities.The probability of an ASR error was, for each episode, a random number in $[0.25, 0.75]$ and the confirmation probability for each episode, a random number in $[0.25, 0.75]$. Dialogue policies trained on high ASR noise have been proven to work on low ASR noise conditions, but not vice versa [150]. We therefore trained the system using a simulator with fairly high ASR noise and tested it on a simulator with lower ASR noise. We opted for a large group of novice users and a small group of experts, as in [125] and thus the simulated user is novice with probability 0.75 and expert with probability 0.25.

4.5.2 Experimental setup

In this section, we describe our experimental setup. To assess AWL's performance, we paired it with a hierarchical SARSA algorithm (AWL-HSARSA) and compared against a simple hierarchical SARSA (HSARSA) and a hierarchical SARSA (HSARSA-h) with a preselected handcrafted set of complex actions (i.e. the optimal complex action set along with the basic actions). We run each algorithm 100 times on a slot filling problem of 3 slots and 6 basic, i.e. 728 complex, actions (3x6) up to 7 slots and 14 basic, i.e. 4,782,968 complex, actions (7x14), due to hardware constraints, allowing for each dimension an increasing number of episodes. It should be noted that we did not allow complex actions composed of other complex actions, so from 6 basic actions, for example, we have $3^6 - 1$ possible complex actions. Since we used hierarchical algorithms, however, this extension is straightforward. Note that a slot filling problem with 8 slots and $(3^{16} - 1) = 43,046,720$ actions has $3^8 = 6,561$ dialogue states and $(3^{16} - 1)^{3^{16}} \simeq 1.34e + 328,616,088$ possible policies. For our HSARSA-h, we used a handcrafted complex action set similar to the actions that [114] used, which are:

- Ask for the value of two slots
- Ask for the value of three slots

Table 4.2. Parameter settings for experiments with varying user simulator behaviour.

Experiments	e_c	ASR_n
E1	1 to 0.9	0 to 0.25
E2	0.95 to 0.8	0.1 to 0.3
E3	0.9 to 0.7	0.15 to 0.35
E4	0.85 to 0.6	0.2 to 0.4
E5	0.8 to 0.5	0.25 to 0.45
E6	0.75 to 0.4	0.3 to 0.5
E7	0.7 to 0.3	0.35 to 0.55
E8	0.65 to 0.2	0.4 to 0.6

- Confirm the value of two slots
- Confirm the value of three slots
- Confirm the value of two slots and ask for the value of another slot
- Confirm the value of one slot and ask for the value of two others

It should be noted that our system does not distinguish between implicit and explicit confirmation actions, as is the case in [114]. Instead, an implicit confirmation would be a complex action composed of a confirmation for the value of a slot and a request for the value of another and an explicit confirmation would be a basic confirmation action. We also let all basic actions be available in the handcrafted set, thus we allow all combinations of basic actions which result in complex actions that are composed of up to three basic ones. For the experiments where we varied both ASR error level (e_c) and the confirmation probability, we evaluated the algorithms in 8 settings, shown in Table 4.2.

To assess the performance of USE, we tested the achievement of a single goal, that varies over time (episodes) and the achievement of two independent goals that also vary over time. Last, we assessed its performance on the achievement of two intertwined goals and two competing goals. More specifically, for a single goal we tested the achievement of:

1. $\tilde{P}(u_t^{(1)} = happy | d_t, a_t)$
2. $\tilde{P}(u_t^{(1)} = upset | d_t, a_t)$
3. $\tilde{P}(u_t^{(1)} = happy | d_t, a_t)$
4. $\tilde{P}(u_t^{(1)} = excited | d_t, a_t)$
5. $\tilde{P}(u_t^{(1)} = tense | d_t, a_t)$

For this experiment we used a 5x5 slot filling problem, without confirmation actions and allowed 150 episodes for each goal to be achieved. For two independent goals we tested the achievement of certain emotional state and attention levels:

1. $\tilde{P}(u_t^{(1)} = happy, u_t^{(2)} = payingAtt | d_t, a_t)$
2. $\tilde{P}(u_t^{(1)} = upset, u_t^{(2)} = notPayingAtt | d_t, a_t)$
3. $\tilde{P}(u_t^{(1)} = happy, u_t^{(2)} = payingAtt | d_t, a_t)$
4. $\tilde{P}(u_t^{(1)} = excited, u_t^{(2)} = notPayingAtt | d_t, a_t)$

Again we allowed 150 episodes in this experiment, for a 5x5 slot filling problem, for each goal to be achieved. In order to perform these experiments, we paired optimal dialogue policies with most possible combinations of goals for emotional states and attention levels. We deliberately did not model all possible cases, in fact some combinations were suboptimal w.r.t $R(s, a)$, that the system had to avoid. We also set some suboptimal policies that achieved the same goals as optimal ones, to see if the system would converge to an optimal policy or not.

A similar experiment was conducted to assess USE's performance, when asked to satisfy competing goals. In this case we assume we have two variables, A and B, where $A \in \{1, ..., 10\}$, $B \in \{1, ..., 5\}$ for which we have that $A = 2B$. We will try to satisfy the following goals:

1. $g_1 : A = 6, B = 3$

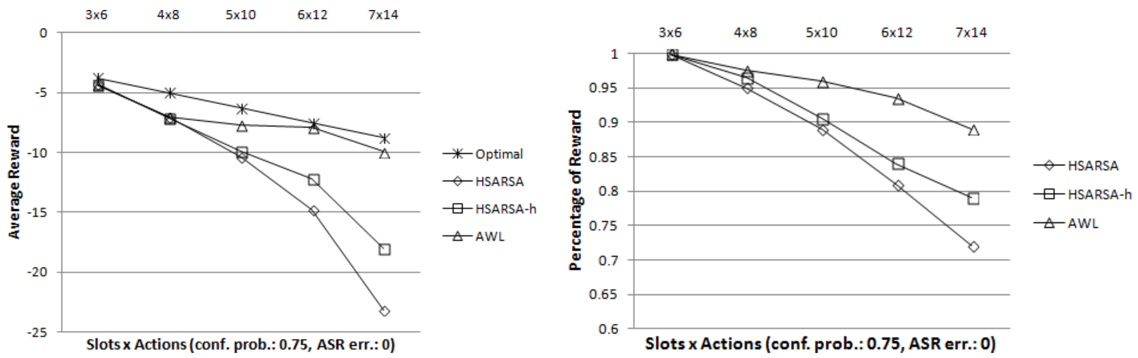
2. $g_2 : A = 8, B = 2$
3. $g_3 : A = 4, B = 3$
4. $g_4 : A = 4, B = 1$

4.5.3 Results

In this section we present the results of our evaluation and attempt to explain them. Figure 4.5(a) shows the average reward each algorithm (HSARSA, HSARSA-h and AWL-HSARSA) achieved, during 100 runs, on slot filling problems of dimension 3x6 (3 *askSlot* and 3 *confSlot* actions) to 7x14, with confirmation actions. We could not test our setup with problems of higher dimensions, because even with application of function approximation techniques, the size of the action space grows exponentially with the number of slots. For specific problems one may find efficient feature sets, but we wanted to test the algorithms on the generic slot filling problem, without any extra domain-related knowledge. Figure 4.5(b) shows the percentage of the available rewards each algorithm achieved on the same problems. In this case the average reward was calculated as:

$$\frac{AvgRew + |-NEpisodes - 99|}{-|q|(2 - e_c) + |-NEpisodes - 99|}$$

Where $AvgRew$ is the average rewards achieved by the algorithms during the test runs and $NEpisodes$ is the number of episodes each algorithm was allowed to run at each test. Note that we cannot really say what the optimal percentage of rewards achieved would be in the scenario with confirmation actions, as we only know the probability of a slot value to be accepted or rejected. In order to calculate the percentage of available rewards then, we estimate the available rewards as $-|q|(1 + (1 - e_c))$, i.e. the number of slots to be filled plus the number of confirmation actions that will be required. This is an approximation, as it only captures the case where roughly

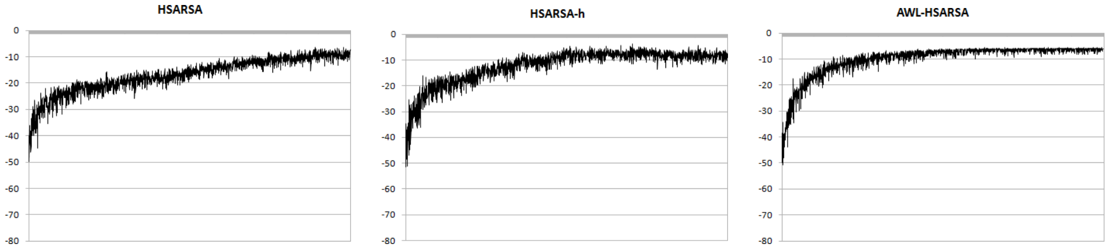


(a) Average Rewards (b) Percentage of Rewards Achieved

Figure 4.5. (a) Average Reward for various problem dimensions, with confirmation actions ($ASR_n = 0$, $e_c = 0.75$). Optimal is achieved by an already trained system. (b) Percentage of the available rewards achieved for various problem dimensions.

$(1 - e_c)\%$ of the slots need to be confirmed and where one confirmation action is enough for each slot. In reality there is also a probability of $(1 - e_c)^2$ that the system will need another confirmation action and so on, but this approximation is enough for our purposes.

In Figure 4.5(a), HSARSA-h seems to perform a little better than simple HSARSA, but AWL-HSARSA again outperforms both algorithms. After dimension 4x8, AWL-HSARSA seems to perform better, possibly due to the fact that it can handle ASR errors well. It should be noted that wherever there is a learning curve, no algorithm can achieve the optimal average reward, which we define in this case as the optimal reward yielded by an already trained system, since during learning, the algorithm will definitely yield some suboptimal rewards. The difference in performance between HSARSA and HSARSA-h in Figure 4.5(a) is that HSARSA needs to learn an optimal set of complex actions and therefore has a longer learning time, yielding lower rewards than HSARSA-h, while learning. AWL-HSARSA on the other hand learns much faster, thus yielding higher rewards earlier on. The average reward achieved is a rough measure of performance, as it provides no way of telling when the



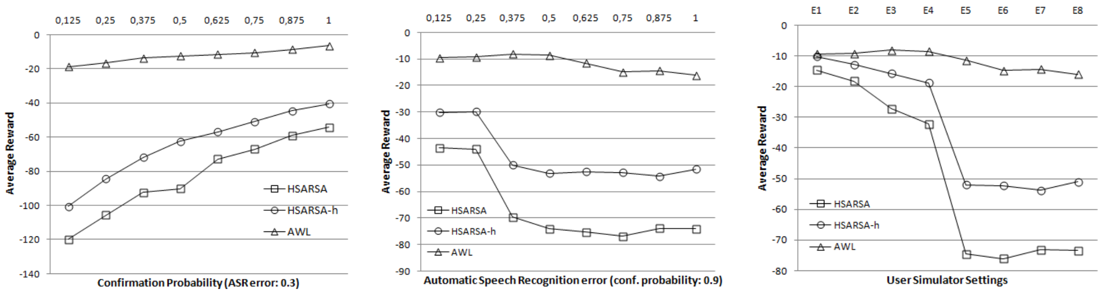
(a) HSARSA (b) HSARSA-h (c) AWL-HSARSA

Figure 4.6. Learning curves on a SFP with confirmation actions (7 slots and 4,782,968 complex actions), averaged over 100 runs.

algorithm actually converged. For example, as in the case of AWL-HSARSA, an algorithm may yield very bad rewards for a very few initial episodes and then converge to the optimal and another may be slower to converge while yielding moderate rewards. Another aspect that affects the average reward plots is the number of allowed episodes, as after the algorithms converge, the average reward achieved can only go towards the optimal. Having a very high number of episodes, though, would prevent a good comparison between the evaluated algorithms. We thus allowed about 30% more episodes than necessary for convergence (for the slowest algorithm), to focus in the interesting part of the learning curves. Figure 4.5(b) shows the percentage of available rewards each algorithm achieved, for each slot filling problem, reflecting the same results.

Figure 4.6 shows the learning curves of each algorithm, on a 7x14 slot filling problem with confirmation actions. We can see that AWL-HSARSA clearly outperforms the other two algorithms, with statistically significant differences.

Figure 4.7(a) shows the average reward each algorithm achieved against varying confirmation probabilities, while ASR error was constantly set at 0.3. It is evident that AWL-HSARSA outperforms the other algorithms by a fair margin, as is the case in Figure 4.7(b), which shows average rewards versus ASR error, while the confirmation

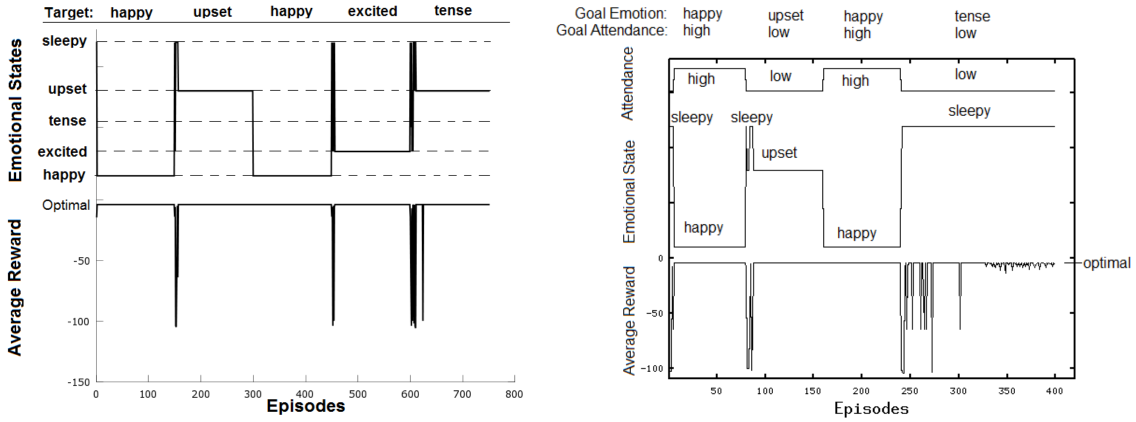


(a) Varying conf. probabilities (b) Varying ASR error (c) Varying both parameters

Figure 4.7. Average reward of HSARSA, HSARSA-h and AWL-HSARSA with varying confirmation probabilities (a), varying ASR error level (b) and varying both parameters (c).

probability was fixed at 0.9. It should be noted here that the drop in performance after $ASR_n = 0.25$ is due to the fact that our threshold of accepting the user's utterance was set to 0.75. This means that for lower ASR_n values the system always accepts the utterance and therefore the error does not affect its performance at all. AWL-HSARSA, however, seems unaffected by the increasing ASR error rate and performs almost the same. Figure 4.7(c) shows the average reward each algorithm achieved in various settings of e_c and ASR_n, according to Table 4.2. In this plot the conditions get worse as the experiments go from E1 to E8. Note here that an ASR_n equal to 1 does not mean that there is 100% chance of an ASR error, it just means that there is up to 100% chance of an ASR error, each time an "askSlot" action is chosen. Also, according to (9), an e_c of 0 means that there is up to 100% chance of an understanding error, i.e. the user rejects the slot value. We can also see that HSARSA and HSARSA-h have a significant drop in performance after E4. AWL-HSARSA again performs better with only a small drop in performance as environmental conditions worsen.

In Figure 4.8(a) we can see the results of the experiments from [2], where USE needs to find an optimal policy that also achieves a single goal (or target). We can clearly see that USE manages to achieve the goals set, when corresponding optimal



(a) USE with a single goal, taken from (Papangelis et al. 2012d)

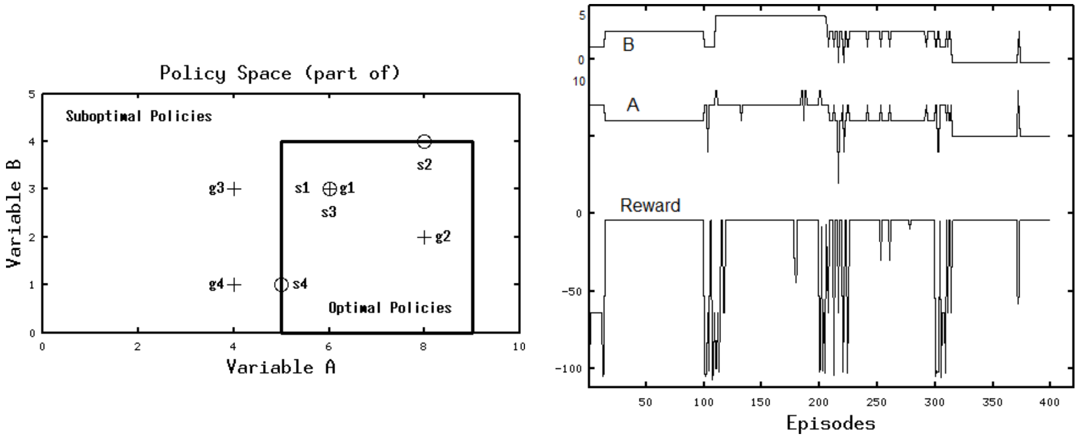
(b) USE with two independent goals

Figure 4.8. Results of USE with one goal (a) and two goals (b) for various changing independent goals.

policies exist, and it finds a policy as close to the goal as possible when not, as in the case when the target emotional state is *tense*. In this case, no optimal policy exists to satisfy this goal, so USE converges to an optimal policy that satisfies a goal which is as close as possible to *tense* and in our modelling this would be *upset*.

Figure 4.8(b) shows similar experiments, but with two independent goals that need to be achieved simultaneously. Note here that we do not make any assumption on the importance of each goal, but the designer of the system may assign appropriate weights to each goal and model their significance. We can see here that USE has no problem achieving both goals, when that is possible. In Figure 4.8(b), we can see that, similarly to Figure 4.8(a), USE manages to achieve both goals, as they change over time. In the first episodes, there is a small learning period before the algorithm converges to an optimal policy that also satisfies the target emotion and attention levels. After episode 80, where the change occurs, there is another learning period and the algorithm again manages to converge to an optimal policy, that also satisfies both targets. After episode 160, there is no learning period as the policy that was learnt in the first 80 episodes has been saved and is now reused. After episode 240, we can see that USE converges to a *sleepy* optimal policy (wrt $R(s, a)$) that also satisfies the *attention* goal. Notice the slight divergence from the optimal rewards after episode 320, which is due to USE attempting to find an optimal solution that will also satisfy the *emotion* constraint. The fact that the algorithm finds suboptimal policies that achieve *emotion=sleepy* and *attention=high* is attributed to the fact that these are the respective default values, as is also the case during the learning period of every target.

It is also interesting to observe USE's behaviour when asked to satisfy competing goals. Of course, the designer may assign weights to convey the importance of each goal, but in our experiments we will consider all weights equal because our purpose is to see if the algorithm will satisfy as many goals as possible. Figure 4.9(a) depicts a schematic projection of a part of the policy space where variables A and B are realised and all goals are marked. The shaded area is the area of that part of the policy space where policies, optimal wrt $R(s, a)$, reside. In order to achieve g_1, therefore, the system need only converge to the corresponding optimal policy. In order to satisfy g_2 as best as possible (as the combination is impossible due to the constraint $A = 2B$), the system must violate $B = 2$, while satisfying $A = 8$ and $A = 2B$. On the other hand, in order to satisfy g_3 as best as possible it needs to violate $A = 4$ and find an optimal policy that satisfies $B = 3$ and $A = 2B$ (i.e. converge to the same policy that satisfies g_1). Last, in order to satisfy g_4 as best as possible, the algorithm needs to violate at least two constraints, either $A = 2B$ and $B = 1$ or $A = 4$ and $B = 1$ while finding an optimal policy that is as close as possible to g_4. As with the previous experiment, we hard coded several policies (optimal and suboptimal) to yield various combinations of A and B. We assigned default values of $A = 7$, $B = 2$ to the rest of the policies.



(a) Part of the policy space

(b) Variables A, B and USE learning curve

Figure 4.9. Results of USE with two variables. Figure (a) depicts part of the policy space, where the rectangle is a projection of the optimal policy space onto the goal space, where variables A and B are realised. Goals are marked as g1, g2, g3 and g4 and final system states are marked as s1, s2, s3 and s4, respectively. Goals are represented as crosses and final states as circles. Figure (b) depicts the learning curve and the yielded values of A and B as episodes progress.

The results are shown in Figure 4.9(a), where we can see that USE guides the system as best as possible. For the first goal, it quickly converges at $A = 6$, $B = 3$, or (6,3) as is also shown in Figure 4.9(b), where we can see that there is a very short learning period (suboptimal rewards). For the second goal, the system converges to (8,4) and satisfies $A = 8$ and $A = 2B$, while violating $B = 2$. Here, Figure 4.9(b) shows a longer learning period, as no policy exists to satisfy g_2, and the algorithm must look for an alternative. The policy found is the best possible solution, given the circumstances and having in mind that the hard constraints must be met at all costs, i.e. the system must converge somewhere inside the *Optimal Policies* square. For the third goal, the system again converges to (6,3), after a relatively long learning period, therefore satisfying the hard constraints (which g_3 violates), $B = 3$ and $A = 2B$ while violating the soft constraint $A = 4$. For the last goal, similarly, the system converges to (5,1) thus satisfying the hard constraints plus the soft constraint $B = 1$ while violating the soft constraints $A = 4$ and $A = 2B$. This is again the best solution, as it is not possible to satisfy more than one soft constraint. Notice, in Figure 4.9(b), that for goals g_2, g_3 and g_4 (which by definition cannot be satisfied) USE yields suboptimal rewards every now and then, while it tries t find a policy that does satisfy the goals. This exploration can be controlled by tuning σ^2. A rule of thumb is that higher σ^2 values lead to less exploration while lower values lead to more attempts to find alternative policies, as also explained in Section 5.

It should be noted here that we tried to achieve the same results using only the HIAC algorithm, with the soft goals incorporated in the reward function:

$$R'(d, a_i) = \begin{cases} -1, \quad \text{if} \quad a_i \neq a_N \\ -10|\{d_i|d_i = 2\}| - 100, \quad \text{if} \quad a_i = a_N, \exists j|q_j = \emptyset \\ -10|\{d_i|d_i = 2\}| - np, \quad \text{if} \quad a_i = a_N, \neg\exists j|q_j = \emptyset \end{cases} \tag{4.5}$$

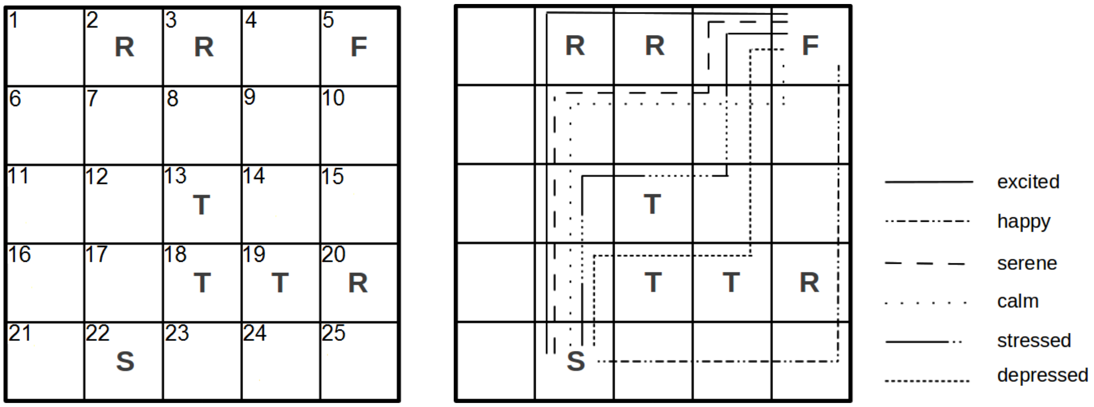
Where n is the number of unachieved optional goals and p is a constant penalty. What we found was that the algorithm is very sensitive to p, as small variations affect the importance of the goals, i.e. the algorithm may converge to dialogue policies that achieve the optional goals and disregard the mandatory ones. Variations in the value of p also greatly affect the number of episodes required for convergence. In short, although one could design a better reward function, we found this approach cumbersome for our purposes, which are to separate mandatory and optional system goals and to be able to switch optional goals in real time, without disturbing the system too much.

4.6 Impact of AWL and USE on the interaction

In this section, we present two example ADS that utilise AWL and USE, aiming to showcase their importance and impact on the interaction experience. The first system is able to provide navigational directions in order to guide the user from a starting point to a desired location, in a world that consists of cells, where at each cell there might be a reward or a trap. The system applies AWL in order to learn how to combine basic direction-giving actions into more complicated ones and USE in order to achieve the hard and soft goals of the system, which are to use the shortest path and retrieve as many rewards as possible, while avoiding traps, respectively. The second example [145], showcases the impact of AWL and is a table top manipulation system, controlled by the ADS, that can perform simple tasks such as picking up objects and placing them somewhere on the table. For this system, we used the Personal Robot 2 (PR2) platform, which has, between others, two robotic arm actuators, cameras, a kinect and a laser sensor that allow us to interpret the scene and identify objects on the table as well as manipulate them. Basic system actions in this case are categorised into *Verbal* and *Physical*, as shown in Table IV. The system can then combine these basic actions into complex ones, able to solve more difficult tasks such as building towers or sorting the objects according to colour. We give a brief overview of the technology we used and we present some example interactions with real users, to showcase the impact and usefulness of the proposed learning algorithms.

4.6.1 Navigation ADS

The ADS for providing navigation instructions is implemented on the Olympus [80] platform, which is a publicly available platform for building dialogue systems. All learning algorithms are implemented in Octave [Octave Community 2012] and communicate with the system via TCP/IP. The architecture of this system is depicted



(a) Grid world with rewards and traps

(b) Paths that evoke specific emotions

Figure 4.10. (a): The grid world, where ‘S’ represents the starting state, ‘F’ represents the goal state, ‘R’ represents states that offer rewards and ‘T’ represents states with traps. (b): Depiction of paths that evoke certain emotional states.

in Figure 5.3, where the AWL and USE modules are implemented in Octave and the links represent TCP/IP connections.

To better showcase the proposed algorithms, we used a scenario where the ADS helps the user navigate from a starting point to a final destination. In our example the world consists of a 5 × 5 grid where the user can be in any of the cells formed (see Figure 4.10). In this world the user can move left, right, forward or backward according to the system’s instructions. Every transition is penalised to encourage navigating the user via the fastest route, but there are cells that offer extra rewards and cells that incur higher penalties. The available system actions are utterances that ask the user to move to some direction or inform the user that s/he has reached his/her destination: *moveLeft, moveRigt, moveForward, moveBackward* and *informEnd*. If the system selects *informEnd* from the destination cell then no reward or penalty is incurred. We can now assume that when the user is guided through cells with extra rewards, his/her emotional state “improves” and when the user is guided through

Table 4.3. Target emotional states and associated dialogue policies, where "L" stands for *moveLeft*, "R" stands for *moveRight*, "F" stands for *moveForward* and "E" stands for *informEnd*

Emotional State	Dialogue policy
excited	F F F F R R R E
happy	R R R F F F F E
calm	F F F R R F R E
serene	F F F R R R F E
upset	L R F F R R F F R E
stressed	F F R R F F R E
depressed	F R R F F F R E
sleepy	all other

cells with extra penalty (traps) his/her emotional state "worsens". The system can thus evoke specific emotional states by guiding the user through the appropriate cells. We let reaching the goal state via the shortest path to be a hard constraint for our system and evoking emotional states to be soft constraints. Extra rewards and penalties therefore only affect the soft goals, which represent evoking specific emotional states. More specifically, Table 4.3 shows the emotional states we wish to evoke and the dialogue policies (navigation instructions) that achieve them. It should be noted here that there are more policies yielding shortest paths (i.e. optimal w.r.t hard goals) and that some of the policies in Table 4.3 do not. This means that those emotional states (e.g. "upset") cannot be achieved and the system will have to evoke a similar emotional state while guiding the user through a shortest path. It should be noted also that there are may be multiple ways of achieving the same emotion (e.g. both "FFFRFRRE" and "RRRFFFFE" include one Reward and thus evoke "happy") and the system may have to choose between them. In Table 4.3 we only list emotions evoked through specific dialogue policies that we are interested in following.

The role of AWL in this system is to decide whether to use simple instructions, such as “move forward” or more complex instructions (resulting from complex actions), such as “move forward, then move left and then move forward”. It should be noted here that a system action could represent *Physical* actions, such as “pick up reward” or “disable trap” and not just navigation instructions that could alternatively be generated using a path finding algorithm and an NLG component. We, however, wish to show how USE can be applied in various settings and learn an optimal dialogue policy that achieves the system’s goals. Moreover, in favour of simplicity, we have not used confirmation actions. In this example ADS there is no real interaction, as the user’s only available action is to acknowledge that s/he has moved to the cell, as instructed. This should not be a concern, however, as this example only aims to better illustrate how USE and AWL can impact the performance and behaviour of an ADS.

In the Appendix we present some sample interactions with the navigation ADS, indicating how USE is able to learn dialogue policies that satisfy our hard constraints, i.e. provide navigation instructions so the user can follow the shortest path, and also satisfy our soft constraints, i.e. evoke a specific emotion. It should be noted that the user’s emotional state is assessed in our simulation after the system has taken the next action, and this is why in the HRL-ADS the first emotional state is “happy” and in USE-ADS it is “calm”. When the system first receives the extra reward, the emotional state of the user changes to “happy” and when it receives another extra reward, it changes to “excited”. The system with USE exhibits similar behaviour with the rest emotional states, as expected, as we can see in Dialogue 3 and 4 in the Appendix. Dialogues 5 and 6 present the output of an interaction with the ADS trained with a simple HRL algorithm and an interaction with the system trained using AWL. In this occasion, AWL was tuned to favour basic actions and complex actions composed of two basic ones. We see that the HRL-ADS uses only simple actions to guide the user, thus requiring more turns. Both systems were trained using the same user simulator, described in Section 6.1, that models expert or novice users and penalises complex actions accordingly. It is evident here that AWL has re-used the complex action {*moveRight, moveForward*}. While in this simple scenario this ability does not impact the system's performance much, we can imagine that when actions represent physical interactions with the environment, database queries, calculations, etc and not just utterances, AWL can speed up the system greatly. In such a case, AWL would combine actions in the most efficient way, and when a task is required again (such as "fetch water from fridge") the system would not need to learn how to achieve it. Moreover, during the interaction, many abstract actions are created leading to the achievement of very complex tasks, such as "take care of patient."

4.6.2 Object Manipulation ADS

The object manipulating ADS is written in C++, using the Robot Operating System (ROS) [151] framework and a Personal Robot 2 (PR2), which was used as our robotic platform. In this ADS we only applied AWL, which was ported into C++ and implemented as a ROS node. The resulting ADS is therefore modular and can be installed in any robotic or other platform that supports ROS. To recognise spoken input, we used the open source pocketSphinx [152] ASR module and to generate speech we used the tts_server package. In this ADS, system actions fall in two categories, *Verbal* and *Physical*, where the first refer to spoken output and the latter to controlling the actuators. The system can, therefore combine basic *Verbal* or *Physical* actions, such as "askColour" or "pickUp" into complex ones, such as "getRedItems".

Table 6.2 presents the basic *Verbal* and *Physical* actions available to the system. Combining these, the system can learn at run time how to solve complex tasks such

Table 4.4. Basic verbal and physical actions, available for our system

Physical Actions	Verbal Actions
PICK	GREET
PLACE	INTRODUCE
OFFER	ASKNAME
	CONFIRM
	ASKTASK
	REPEAT
	END

as stacking, building, sorting, etc. An advantage of using AWL is that the system will learn how to achieve a task, taking into account the user's preferences, meaning that for different users it will learn different ways to achieve the same task. This, of course, assumes that the user is somehow involved in the rewarding process (providing feedback to the system), which can be done, for example, by having a vocabulary of positive and negative expressions, that are used appropriately to express desired or undesired actions taken by the ADS.

More specifically, we define the SFP in this case as follows. The slots to be filled are defined as $Z =< object, colour, a_u >$, where *object* is a unique identifier of the object that the user needs to manipulate, colour is the colour of that object and a_u is the action the users want to perform with/on that object. Slots can take the following values: $object \in \mathbb{N}^0$, where 0 denotes any object (the user may want any red block instead of a particular red block), $colour \in \{red, yellow, blue\}$ and $a_u \in$ {GREET, PROVIDE-NAME, PROVIDE-COLOUR, PROVIDE-OBJECT, PICK, PLACE, OFFER, SELECT, SORT, STACK, BUILD, END}. The system actions, as mentioned before, are presented in Table IV and can be either *Verbal*, *Physical* or complex actions formed by combinations of the basic ones. We define the dialogue state as a vector that contains all the necessary information for this system

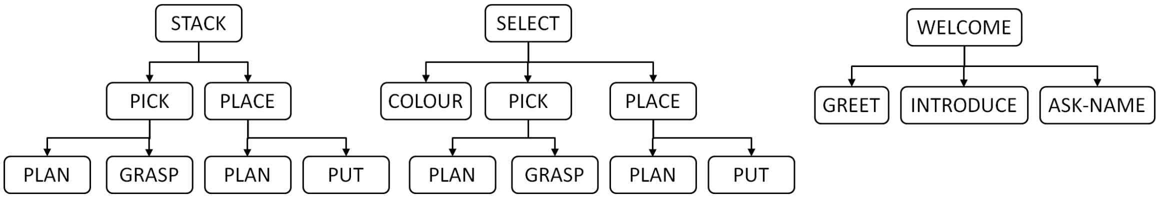


Figure 4.11. Complex actions that can be generated by AWL for the object manipulation system.

to operate: $d_s =< history, object, colour, a_s >$, where $history = \{d_{s,i-1}, a_{u,i-1}\}$ contains the previous dialogue state and user action. To learn the best way to achieve a task such as sorting the objects according to colour, we apply AWL which is able to learn actions similar to those presented in Figure 4.11. A complex action need not comprise solely of *Physical* or *Verbal* actions, but can be a combination of both. We represent complex actions as trees, which are generated and modified at runtime by the system.

The advantage of using an automated complex action learning technique is that the system will figure out the best way (according to the reward function) to combine basic or complex actions into more complex ones and solve complicated tasks, such as sort objects according to colour. Such a task could, for example, include greeting the user, retrieving the user's intentions, and executing a “sort” action, for each available colour. A “sort” action, in turn, would comprise many “select” actions, that pick up an object of a specified colour and place it in a specified location. The designer, therefore, does not need to manually create complex actions, which are not only hard to design but also to maintain. Moreover, adaptation is easy to achieve if we use a complex action learning technique, as the system will not only select which action is appropriate to use, depending on the circumstances, but also change the way to achieve a task, by altering an existing complex action or generating new ones. Figure

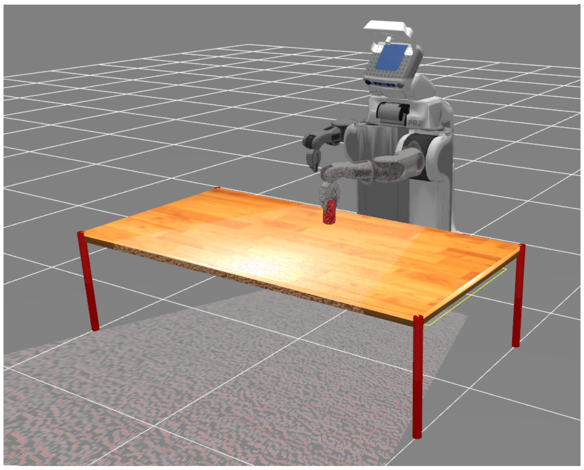


Figure 4.12. Simulation of PR2 manipulating an object.

4.12 depicts a simulation of our system running on a PR2 in Gazebo, having picked up an object.

4.7 Discussion

We have presented two batch learning algorithms for achieving adaptation in DS, specifically for learning how to combine simple actions into more complex ones and for learning how to guide the system towards achieving its goals. Both algorithms have been introduced in [2] and were extended, thoroughly evaluated and presented in detail in the present work. These extensions include automatically estimating how many complex actions are necessary for the ADS. We also presented two real ADS where those algorithms were applied and discussed the algorithms' impact through extensive evaluation and example interactions. In order to better evaluate AWL and USE, we also extended our user simulation model to be able to model ASR errors, slot value confirmation and to yield multiple metrics (such as emotional state or attention level) and we also allowed some parameters, such as ASR, confirmation probabilities or user expertise, to vary randomly but within specified bounds, to avoid overfitting the system to the user simulator (or to a specific setting of its parameters).

These algorithms are a step in the direction of making *natural conversation* with a DS a reality, as they enable the system to take part in more complicated dialogues, with experienced users and also make it flexible in achieving its own or the user's goals, despite changes in the environment. An obvious step forward would be to apply these algorithms in order to estimate the user's behaviour, response or reaction to certain circumstances. This will help the system plan its own actions better and appear as an intelligent interlocutor. We are currently in the process of evaluating the system using two robotic platforms (PR2 and PeopleBot). This will allow us to get a better understanding of the weaknesses and strengths of the presented algorithms, when applied in a real world environment. This will also allow us, in the future, to evaluate the capabilities of the system when operating in an assistive living environment.

In the future we plan to apply the proposed algorithms to error handling and recovery in ADS. We will examine the case where we measure success by assigning a reward after each episode depending on how many errors (from those occurred) were resolved and how many turns it took: $errors \cdot (AvgTurns)^{-1}$. An error handler could be triggered when our confidence about something (such as a slot value or the user's intentions) is low. Moreover, we plan to focus on continuous reinforcement learning techniques as a possible way to handle the scalability problem and see how the presented algorithms deal with continuous space problems. We also plan to implement a stochastic reward function to model the different perception of the system by different users and also other unforeseen aspects of the environment. In the same context we will evaluate the performance of our system when it is trained using stochastic policies. Last, we wish to extend AWL to further prune the action space and remove or group actions that have similar effects [153, 90], as they will probably receive similar ranking.

In the next Chapter, we present several ADS that implement algorithms and models presented in this thesis, for various applications.

Algorithm 5: Action Weights Learning with automatic k estimation

```
Input: Z, A, Number of episodes, k.
Output: A, sorted according to actions' performance.
Initialize k, W, HRL parameters
for $i$ = $episode$ to $NEpisodes$ do
    $\{A_T, reward\} \leftarrow$ HRL(W,k,...);
    $S_{A_T} \leftarrow S_{A_T} + A_T$;
    forall $\{t|A_T(t) = 1\}$ do
        $W(t) \leftarrow \frac{(S_{A_T}(t) - \mathbf{1})W(t) + |reward|}{S_{A_T}(t)}$
    end
    $P_a(a|s) \leftarrow 1 - \frac{W}{max\{W\}} + 0.001$
end
$W_{sort} \leftarrow sort(W, \text{“}descending\text{”})$;
$Ind \leftarrow indexOf(W_{sort})$;
//Automatically estimate k:
$AvgDiff \leftarrow |W_{sort}(1) - W_{sort}(2)|$;
for $i = 2$ to $|W| - 1$ do
    $diff \leftarrow |W_{sort}(i) - W_{sort}(i+1)|$;
    if $diff > 2 \cdot AvgDiff$ then
        $k \leftarrow i$;
        break; ;
    end
    $AvgDiff \leftarrow \frac{i \cdot AvgDiff + diff}{i+1}$
end
return $A(Ind(i)), i \leq k$
```

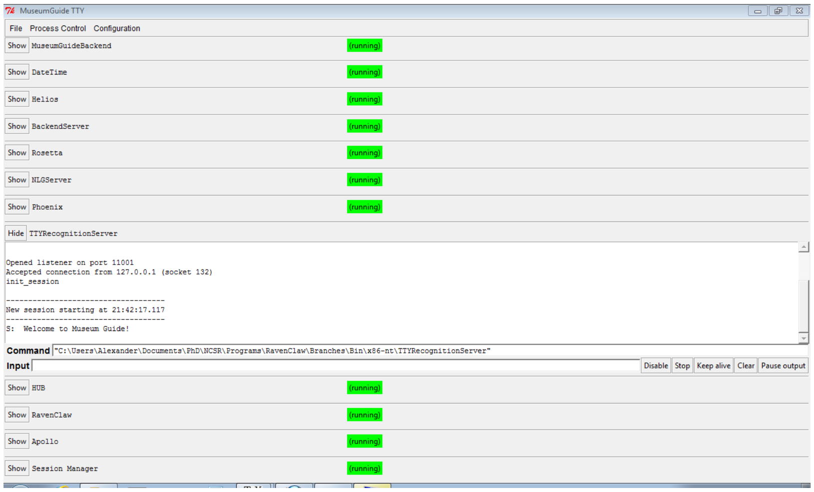
CHAPTER 5

Applications of Adaptive Dialogue Systems in Assistive Living Environments

In this Chapter, we present some applications of our proposed algorithms, to showcase their impact on each application and to emphasize the broad potential of ADS, especially in ALE. Combined, these systems form the core components of our Robotic Assistant for ALE. Last, we present an ADS development framework, targeted for fast and easy development of ADS, even from non-experts, and supports the most prominent DM theories.

5.1 Museum Guide

This system was the product of our evaluation studies on online RL methods for dialogue policy learning. We created a virtual museum guide ADS, based on the IS formulation, that prompts the user for item characteristics (such as era, exhibit type, etc.) and responds accordingly. If there are less than 3 hits in the database of items, the system automatically retrieves their description, otherwise it keeps asking for more information. It should be noted here, that knowing when an item is not found and informing the user, is also a correct system action. This system was developed in C++ and Octave, using the Olympus / RavenClaw [80] framework, and has interchangeable learning modules. To achieve interchangeability, we developed a common data structure to represent dialogue policies, so a policy learned by an algorithm can be used and modified by another. Figure 5.1 depicts a screen-shot of this system.



MuseumGuideBackend (running)
DateTime (running)
Helios (running)
BackendServer (running)
Rosetta (running)
NLGServer (running)
Phoenix (running)
TTYRecognitionServer

```
Opened listener on port 11001
Accepted connection from 127.0.0.1 (socket 132)
init_session

------------------------------------
New session starting at 21:42:17.117
------------------------------------
S:  Welcome to Museum Guide!
```

Command `"C:\Users\Alexander\Documents\PhD\NCSR\Programs\RavenClaw\Branches\Bin\x86-nt\TTYRecognitionServer"`

Input

HUB (running)
RavenClaw (running)
Apollo (running)
Session Manager (running)

Figure 5.1. Screenshot of the Museum Guide ADS.

Our main contribution with this system, is that we propose an architecture for ADS that applies several online RL algorithms for dialogue policy learning that are simple to implement and of low cost to run. Such a system is able to learn policies from simulated users for several user categories and these policies are then used as initial policies when the system interacts with a real user. One can also provide handcrafted policies when expert knowledge is available. The system is therefore able to rapidly adapt to a new user since it exploits prior knowledge and continues to learn throughout the interaction. In the next sections we review the design of our system, present our evaluation and discuss the results.

5.1.1 System Overview

Our system is designed to act as a museum guide and provide descriptions of exhibits in a virtual museum. It is based on the INDIGO [154] system and was implemented using the Olympus [80] platform, which is a platform for developing dialogue systems for research purposes. INDIGO is an affective museum guide dialogue system, able to adapt to different user personalities and user expertise levels. It can

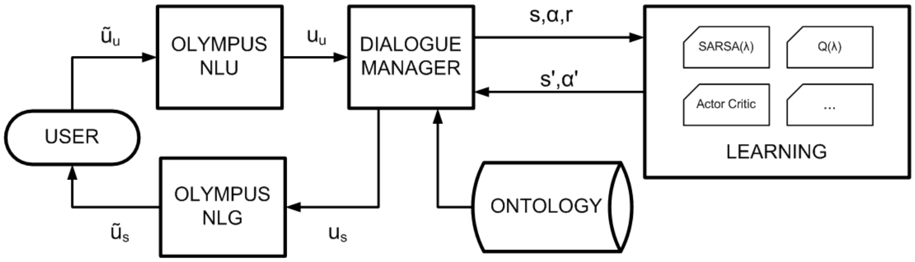


Figure 5.2. Architecture of the proposed Museum Guide System.

assess the user's mood and emotional state and adapt its output accordingly. For dialogue management we used Olympus' RavenClaw Dialogue Manager (DM) and extended it with online RL modules.

Figure 5.2 depicts the architecture of the proposed Museum Guide dialogue system, where s, a, r are the current state, previous action and reward received respectively and s', a' are the new state and new action. $\tilde{u}_u$ and $\tilde{u}_s$ are the noisy user and system utterances while u_u and u_s are the interpreted user utterance and actual system utterance respectively. The system uses Olympus' NLU and NLG components and also has a learning component, where all learning algorithms are implemented. The DM receives u_u which contains the reward of the last action r. It then sends s, a and r to the learning component and receives back the new system state s' and the new action a' it should take. It then takes action a' and sends a description of the system's utterance u_s to the NLG component. Last, it has an ontology that is based on INDIGO's ontology, which describes museum artefacts, time periods, persons and more and contains information about many exhibits. Museum Guide currently supports one type of query and that is requesting for a description of an exhibit. Search in the ontology is performed by providing values for the exhibit's type, construction time, or time period if it is a person and its physical location.

To achieve adaptation in ADS researchers should carefully select methods that are able to tackle the many challenges of this field. RL can successfully handle many of those, such as error recovery and robustness to environmental changes. More specifically, RL can be applied in Dialogue Management (DM) in order to find an optimal dialogue policy that will yield the best action the system should take, depending on the state it is in. Using online RL techniques, the system is able to learn continuously and adapt to changes or different users.

We have implemented a variety of online RL algorithms covering a broad range of the available methods. Our system therefore is able to continue learning as it interacts with real users as well as switch learning methods at will, depending on the problem at hand. Each algorithm's output is in a standardized form and so policies are interchangeable, meaning we can learn a policy using one algorithm and apply it using another. This gives the designer the option of providing a handcrafted policy (modeling prior knowledge) to the system and the system will then optimize that policy according to its current needs. One can also have several user categories and provide handcrafted policies or policies learned through simulations, for each category. The system will be able to use them as initial policies when interacting with appropriate users and refine them to adapt to the specific users' needs. We will now briefly describe the algorithms implemented in our system.

SARSA(λ) (S(λ)) is a very popular RL algorithm, often used in ADS. It applies temporal difference methods for learning and produces an estimate of $Q^{\pi}(s, a)$, represented as a matrix [144]. λ is a parameter thar controls how much effect will past experience have in future updates. SARSA(λ) is a model-free algorithm, but we also have a model-based version implemented, called DynaSARSA(λ) (DS(λ)).

Q-Learning (Q) was proposed by Watkins [141] and it also learns an estimate of $Q^{\pi}(s, a)$. The major differences with SARSA(λ) are that it calculates the difference between $Q^\pi(s', a^*)$ and $Q^\pi(s, a)$, where s' is the new state and a^* is the greedily optimal action, that it only updates a single entry of the matrix representing $Q^\pi(s, a)$ at each iteration and does not use eligibility traces, which is a matrix that keeps track of states visited and actions taken. We also have a Dyna version implemented for this algorithm (DQ).

Q(λ) is similar to Q-Learning in the sense that it uses the temporal difference between $Q^\pi(s', a^*)$ and $Q^\pi(s, a)$ and similar to SARSA(λ) in the sense that it updates the whole $Q^\pi(s, a)$ matrix at each iteration and uses eligibility traces [141, 155]. There is a Dyna version for Q(λ) as well (DQ(λ)).

Actor Critic QV (AC) is an algorithm that uses an actor to follow the policy that is being learned and a critic to evaluate it. This algorithm uses the temporal difference between $Q^\pi(s, a)$ and $V^\pi(s)$ to compute updates for the actor and the critic [156]. We have also implemented a model-based version for this algorithm (DAC).

Our system can operate in two modes. Either in *simulation mode*, where the learning algorithms are trained using simulated users, or in *real user mode* where the system interacts with real users while continuing to learn at a lower pace. A policy learnt using either mode can be saved and loaded for later use again in either mode. During training the system interacts with a real or simulated user and receives a reward after each action (i.e. system utterance). Through this process the system learns which actions bring higher rewards and in what order they should be taken. At the initial stages of training, RL algorithms explore the state-action space and, therefore, make many mistakes. Below we can see example dialogues before training, where actions are taken in a near random way and after training with SARSA(λ):

Dialogue 1: Untrained System

S: Welcome to Museum Guide!

S: What is the location you are looking for?

U: In Athens

S: What is it you'd like to learn about?

U: An altar

S: What is the time period you are looking for?

U: Around 100bc

S: What is the time period you are looking for?

U: Around 100bc

S: What is the location you are looking for?

U: In Athens

S: What is the time period you are looking for?

U: Around 100bc

S: What is the location you are looking for?

U: In Athens

S: Welcome to Museum Guide!

⋮ (omitted)

S: [ITEM DESCRIPTION]

S: You can say, start a new query, or goodbye.

U: bye!

S: Thank you for using Museum Guide, goodbye!

Dialogue 2: Trained System

S: Welcome to Museum Guide!

S: What is it you'd like to learn about?

U: An altar

S: What is the time period you are looking for?

U: Around 100bc

S: What is the location you are looking for?

U: In Athens

S: [ITEM DESCRIPTION]

S: You can say, start a new query, or goodbye.

U: bye!

S: Thank you for using Museum Guide, goodbye!

5.1.2 Evaluation

Evaluation of ADS or DS in general is still an open question. Many approaches and methods have been proposed as listed in detail in [157]. When researchers use RL methods the most common approach is to use the reward function $R(s, a)$ as a metric of performance, since it is explicitly defined to minimize or maximize standard metrics such as dialogue length or goal achievement. In our experiments we evaluated each algorithm's speed of convergence, using the reward function $R(d, a)$ as defined later in equation (1). In the rest of this section we will formally model the dialogue problem, describe our experimental setup and present the results of our evaluation,

Table 5.1. Average learning speed

Alg.	*Conv. Ep.*
S(λ)	**8.52**
Q	13.8
Q(λ)	12.24
AC	11.6
DS(λ)	31.32
DQ	12.16
DQ(λ)	26.6
DAC	29.04

where we focused on the comparison of the different techniques. For simplicity, we opted for a noise free scenario.

The dialogue problem can be formulated as a *slot filling problem.* In our evaluation our problem had 3 slots, *Type, TimePeriod* and *Location*, and typically 6 actions, *Welcome, AskType, AskTimePeriod, AskLocation, GreetGoodbye* and *GiveAnswer*, out of which we hardcoded the first action to always be *Welcome* and the last to be *GreetGoodbye*. The system then needed to learn how to retrieve the three slots in the most efficient way. We evaluated the system using a simple noise-free user simulator that always responds correctly to any system request. An episode in this problem is over when the system presents the results to the user (which may or may not be the right thing to do). For each algorithm we counted the number of episodes it took to learn the optimal policy and averaged it over 25 runs. Note here that convergence speed is correlated with the average total reward, since the sooner the algorithm converges the higher the reward (it will be performing optimally from the point of convergence and after). The results are shown in Table 1, below.

As we can see in Table 5.1, SARSA(λ) outperforms all algorithms while enjoying statistically significant differences, as shown in Table 5.2, and DynaQ Learning

Table 5.2. Statistical significance, where $\star$ denotes $p < 0.0001$

Alg.	S(λ)	Q	Q(λ)	AC	DS	DQ	DQ(λ)	DAC
S(λ)	1							
Q	$\star$	1						
Q(λ)	$\star$	**0.0486**	1					
AC	**0.0028**	**0.0352**	0.4961	1				
DS(λ)	$\star$	$\star$	$\star$	$\star$	1			
DQ	**0.0032**	0.1804	0.9443	0.6724	$\star$	1		
DQ(λ)	$\star$	$\star$	$\star$	$\star$	0.1950	$\star$	1	
DAC	$\star$	$\star$	$\star$	$\star$	0.6120	$\star$	0.5355	1

outperforms the rest model based algorithms. SARSA(λ)'s performance can be in part explained by the eligibility traces that allow past experience to aid the learning process. Model based algorithms evidently do not perform that well, with the statistically insignificant differences (with $p > 0.05$) explained by the fact that these algorithms (including AC) are highly unstable and therefore have very high variance in terms of number of episodes required to converge. We implemented all algorithms in our system to allow the designer freedom of choice (each algorithm is representative of an RL class of methods) and flexibility in unforeseen issues.

As we can see from Dialogue 1 and 2, there is a clear improvement on the system's behaviour during learning and the optimal dialogue policy resembles one that a human designer would use. While this is a very simple scenario, one can imagine that in a system with many more slots and admissible queries, handcrafted policies are very hard or even impossible to create and inflexible when coming to adapting to users' needs and to abrupt changes in their goals. Online RL can deal with such problems and scale to real world applications thus alleviating the need of huge and complicated handcrafted policies. As mentioned before, our system provides the option of importing a handcrafted policy, that represents prior knowledge that the system may refine to its current needs.

In the future we plan to implement state of the art online RL algorithms, such as Natural Actor Belief Critic [158]. We also plan to apply hierarchical RL to achieve a more natural representation of the system's available actions and the users' goals and also apply techniques such as Complex Action Learning. To this aim we will need to formulate the slot filling problem as a Semi Markov Decision Process (SMDP). SMDPs allow for temporal abstraction in the model, meaning an action can take an arbitrary amount of time to complete and so we can model complex actions. Last we plan to test the Museum Guide system with real users to gain valuable feedback and intuition.

5.2 Navigation

This system was developed for evaluation purposes, and specifically to evaluate AWL and USE. It builds on top of the museum guide system, so it has interchangeable learning modules, with the addition of IAC, NAC, AWL and USE. The system is designed to operate on a grid world, where, for example, a system utterance 'please move forward' means move to the next 'block'. This system is also implemented using C++ and Octave on the Olympus / RavenClaw platform. This system currently has an interface similar to the Museum Guide ADS and was presented in more details in the previous Chapter, Section 4.5.1.

5.3 Object Manipulation

The greatest challenge of ALE is that each case (user and environment) is different and therefore it is hard to develop a system that works in every possible scenario (elderly, disabled, injured patients, wounded warriors, etc). If we want assistive technology to be widely accepted, therefore, it is imperative to be able to seamlessly adapt to different users and possibly dynamic environments. To this end we incorporate complex action learning in an ADS, which enables the system to learn, in an online fashion, different solutions to the same problems, depending on the feedback it receives from the user and the environment. The system interacts with users using spoken natural language, instead of simple vocal commands, and can store user profiles, containing information about previous interactions as well as user preferences, especially on specific ways to achieve certain tasks. Online learning guarantees that the system will continue to adapt and therefore if a user's preferences change, the system will adapt quickly. While our use case is quite simple (manipulating objects on a table) compared to real user needs (preparing meals, washing dishes, etc), one can imagine the spectrum of possible applications. In the following section we present some of the necessary background knowledge, in section 3 we present an overview of our system, in section 4 we describe our ADS in detail and in section 5 we conclude and present our plans for future work.

5.3.1 System Overview

Having in mind the challenges and the current state of the art, we present an assistive object manipulation system that is controlled by an adaptive dialogue system (ADS). ADS are systems that interact with their users in a natural manner, typically accepting input from various modalities, such as speech, text, facial expressions, gestures, etc, and responding again in a multimodal fashion. Such systems are usually designed to perform a specific task, such as customer support or flight booking, and in our case we designed an ADS to aid users in tabletop manipulation tasks. Our system is able to understand the user's intentions and control a robot that is able to manipulate objects on a table, thus performing tasks such as building towers of objects, sorting the objects according to colour, offering an object to the user etc. The user interacts with the robot via an ADS that is able to adapt to each user and to new environments and is also able to combine actions that solve simple tasks into complex actions that solve more complicated tasks. Our system is, therefore, able to appear more intelligent, by exhibit complex learning behaviour and the user can thus trust the system more easily.

Figure 5.3 depicts the architecture of our system, which accepts input from the environment and the user, and reacts accordingly. The user's input in our case is speech, that is captured by an Automatic Speech Recognition (ASR) module [152] which produces a textual form of the user's utterance. This is passed on to a Natural Language Understanding (NLU) module that analyses it and identifies the user's intentions. The state of the environment (objects in the table, their colours, etc) is captured by various sensors (stereo camera, kinect) and this information is passed on to a Scene Interpretation module, that understands which objects are on the table, what colors they have, if they are graspable, etc. Output from this module and the NLU is then passed to the Dialogue Manager (DM), that is responsible for fusing the information and making a decision on how to react. The system may react in two forms, either say something or do something, i.e. take a *Verbal* action or a *Physical* action. If the system decides to take a verbal action, it generates the appropriate output, that conveys the system's intentions (e.g., retrieve information about placement location) and passes it to the Natural Language Generation (NLG) component that translates it into text. The text is subsequently passed on to the Text To Speech (TTS) component that generates spoken output. If the DM decides to take a physical action, it generates the appropriate command that is sent to the Motor Control module which controls the robot's actuators. In the following sections

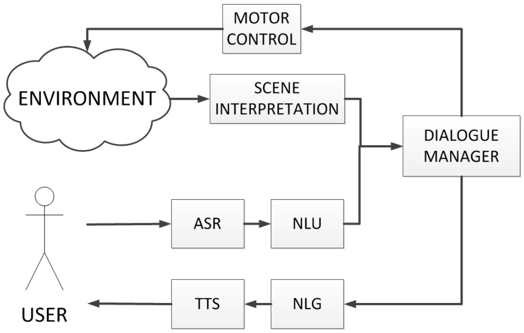


Figure 5.3. The architecture of our system.

we provide more details on the ADS and how we are able to successfully manipulate objects.

5.3.2 Adaptive Dialogue System

In this section we describe our ADS, whose purpose is to interact with the user, make sense of the information contained in the user's input and react accordingly. The system is not only able to understand the user's needs but also to learn how to meet them, by combining basic system actions, such as picking up an object and placing an object somewhere, into more complex ones such as stacking objects or sorting objects. It is therefore able to adapt to each user's preferred way of achieving a task. In the following sections we first provide some theoretical background, we then present our ADS in detail and talk about how we achieve online complex action learning.

To model the dialogue problem, we use the model proposed in [145], which is based on the *Slot Filling / Information Seeking* paradigm. In our environment the user may perform one of the following actions: $u_a \in$ {GREET, PROVIDE-NAME, PROVIDE-COLOUR, PROVIDE-OBJECT, PROVIDE-ACTION}. We define the following slots: <object, colour, u_a > that fully grasp the intentions of the user.

Table 5.3. Basic actions available for our ADS, out of which complex actions will be generated.

Physical Actions	Verbal Actions
PICK	GREET
PLACE	INTRODUCE
OFFER	ASKNAME
	CONFIRM
	ASKTASK
	REPEAT
	END

More specifically, *object* is an object identifier (an integer) and can take values from $\mathbb{N}$, *colour* can take values from {red, green, blue, yellow, white, black} and *action* can take values from {pick, place, offer, select, sort, stack, build}. It should be noted here that these correspond to actions that the user wants to be performed. The dialogue state of our system is defined as a vector: $d =$<history, object, colour>, where *history* comprises the previous dialogue state and the previous user action: $history = \{s_{i-1}, u_{a,i-1}\}$.

In our system, actions are split in two categories: *Physical* and *Verbal*. As the names suggest, physical actions correspond to moving the robot's actuators while verbal actions correspond to the robot saying something. It should be noted that in general other types of actions may be defined, such as database retrieval or computational actions that do not fall in the these two categories, but we will not deal with those in this work.

Another characteristic of system actions is that they can be basic or complex. Basic actions are simple actions that the robot can execute, such as "grasp", "move", "greet", "askColour", etc. Complex actions correspond to more complicated tasks and are composed of basic or other complex actions, for example the complex action “sortItems” would comprise many pick and place actions, “getUserIntentions” would comprise many verbal actions, etc. We use a hierarchical data structure to represent complex actions and apply hierarchical RL algorithms to find an optimal dialogue policy. In a typical SFP DS, most *Verbal* system actions correspond to prompts for slot values (e.g. “Which object would you like?”) but in our case an action may be a combination of several verbal actions and physical actions, such as “askColourStackItems”, which would stack the objects of a specific colour. While our scenario is quite simple, one can imagine how *Verbal* and *Physical* actions can be intertwined when the robot is, for example, asked to prepare lunch. Table 5.3 presents the basic actions of our system, out of which complex actions can be automatically generated.

In order to learn how to combine basic actions into complex ones and solve complicated tasks (such as to build a tower of blocks, or sort the blocks according to colour), we use the Action Weights Learning (AWL) algorithm [2]. This algorithm is able to learn complex actions such as the ones depicted in Figure 5.4, where SELECT is composed of asking for a colour, and then picking and placing every object of that colour. BUILD is composed of asking for a shape (e.g., a square), and then stacking some objects, where STACK is composed of a PICK and a PLACE action. SORT is composed of one SELECT action for each colour and WELCOME is composed of greeting the user, introducing the robot and asking for the user’s name. Complex actions are internally represented as trees, that are generated and modified at runtime. Basic actions are located at the leaves of each tree and internal nodes represent complex actions composed of their children. In order to execute a complex action, therefore, the system must recursively execute its children actions.

The advantage of using an automated complex action learning technique is that the system will figure out the best way, according to the reward function, to combine basic or complex actions into more complex ones and solve complicated tasks. The

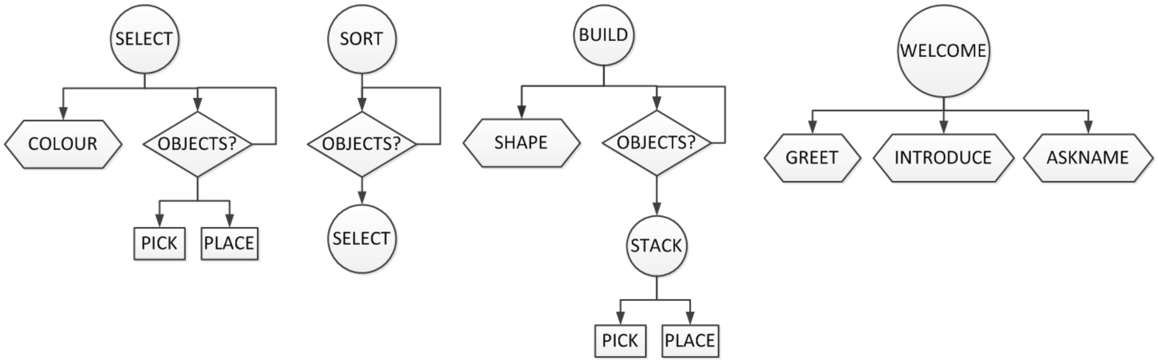


Figure 5.4. Sample complex actions our system is able to learn. Squares denote physical actions, hexagons denote verbal actions and circles denote complex actions. Rhombuses are only used in this figure in favour of compactness.

designer, therefore, does not need to manually define solutions to complicated tasks, which are not only hard to design but also to maintain. Moreover, adaptation is easy to achieve if we use an online complex action learning technique, as the system will not only select an appropriate action, depending on the circumstances, but also change the way to achieve a task, by altering an existing complex action or generating new ones.

The Dialogue Manager, depicted in Figure 5.5, is the central component of the ADS and is responsible for deciding how the system should react, depending on the current user, the environment, the system's goals, etc. It receives input from the NLU component, which provides information about the user's utterance and the Scene Interpretation component which provides information about the objects in front of the robot, such as colour or location. It then processes this information and computes the current state of the dialogue d, taking into account the state of the world and the progress of the interaction so far. The user action u_a is derived from the user's utterance and passed on to the learning module, along with the current dialogue state. The learning module implements the AWL and HSARSA algorithms and computes

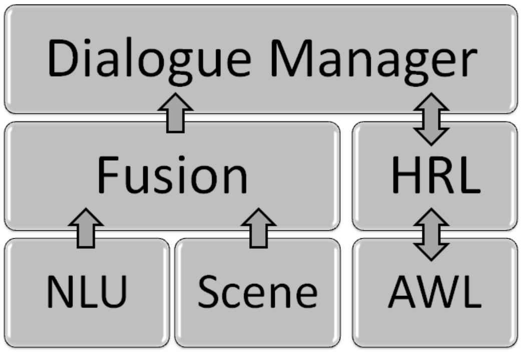


Figure 5.5. The architecture of the dialogue manager.

the response of the system (i.e. the next system action). The learning module also makes the necessary updates, as both algorithms are online. After receiving the next system action, it either forwards it to the NLG component or to the Motor Control component, depending on whether it is *Verbal* or *Physical*.

We used the Personal Robot 2 (PR2) as our evaluation platform, which is a robot equipped, among others, with two arms and several sensors, such as kinect, stereo cameras, laser, etc. In order to manipulate objects, therefore, we used some the available open source modules [151] for scene interpretation and object manipulation. To interpret the scene on the table and identify objects (colour, orientation, shape, etc.), we use packages such as the *tabletop-object-perception*, which receives input from the stereo camera and the kinect sensor and produces a list of graspable objects. According to the user's intentions then, we can manipulate these objects accordingly, utilising the planning and control algorithms provided by the community.

5.3.3 Discussion

In the present work we present a system that is able to communicate with the user in a natural and intuitive way, understand his/her intentions and successfully manipulate objects on a table. To achieve this, we employ an ADS with online complex action learning, that allows the system to learn how to achieve complicated tasks by reusing previous knowledge. We implemented our system in Robot Operating System [151] and used the PR2 as our robotic platform. The benefits of our approach are that the system can easily adapt to various environments and different users without the need of additional training and fine tuning. Moreover, being implemented in ROS allows easy application on different robotic platforms with relatively little effort.

In order to train the ADS for our application we developed user simulator that produced synthetic data. It is important to have a realistic user simulator, as that is what the system will be trained with, but it is as important to take care not to over-fit the simulator. We attempt avoid this by adding some randomness in our simulator. While there are concerns on using user simulators for training rather than real dialogue data, there are some benefits such as the fact that we can generate a virtually unlimited amount of training data, which we can regulate and parametrise. Also, since our system will apply online RL techniques we only need to find a good enough dialogue policy, where the system will start from, and then as users interact with the system this policy will be fine tuned. In the future we plan to extensively evaluate our system, to add spatial language understanding and experiment with various other objects and more challenging settings. Last, we plan to investigate simultaneous execution of multiple actions, in a manner similar to [62].

5.4 Adaptive Dialogue Management Development Framework

In this section we present our approach to an Adaptive Dialogue Management development framework, in C++ and ROS. In order to capture the most prominent DM models and theories, we propose a new data structure, a Partially Observable Semi Markov Decision Process (POSMDP), out of which all other necessary models can be derived, specifically POMDP, SMDP, HMDP, MDP and FSM, that have been used in the literature. Our aim is to allow non-experts in dialogue management or programming languages to be able to use our framework and, to this end, we will develop a smart drag-and-drop interface, easy to use by novice users.

Dialogue managers (DM) play a central role in many human-computer interaction applications of language technology, maintaining information about the current dialogue state and orchestrating multiple interaction components so that user acts are interpreted and reacted to. The complexity of the DM's task varies greatly, from managing relatively simple voice portals to deliberating about how to coordinate complex cognitive systems capable of multi-modal sensing and actuation. Furthermore, and besides their more practical applications, DMs are also vehicles upon which *discourse theories* can be implemented. In this multi-dimensional situation, the information structures and primitive operators defined in DMs are determined and conditioned by the *dialogue management* framework that the DM adheres to; the *discourse theory* that the DM implements; and the host of linguistic and other *capabilities* available to the system for interacting with the environment. With the growing significance of *adaptive dialogue systems (ADS)*, and statistical and machine learning methods in general in computational linguistics, a fourth dimension has become increasingly relevant: the ways in which adaptivity methodologies interact with the underlying frameworks, theories, and systems.

Reviewing the state of the art, however, reveals that there is no clear way to separate system behaviours that are specifically required by the application and should *not* be subject to adaptivity and those behaviours (or aspects thereof) that should. In this paper we present a dialogue management architecture and implementation infrastructure where the DM model explicitly comprises (a) application and usage-invariant theories of interaction and of discourse; (b) usage-invariant behaviours required by each application; and (c) behaviours that dynamically adapt as the system is used. Our architecture allows developing general-purpose adaptable system behaviours and capabilities that are not tied to a particular DM.

A *Dialogue Manager (DM)* is typically perceived as a state machine where at each point in the interaction, the current state represents all information that needs to be retained from the interaction history. An operator combines user input or other stimuli from the environment with the current state, and computes the system's response and the next state. In this section we briefly review dialogue management approaches from the perspective of the data structures maintained in their states and the operators defined over them.

5.4.1 State transitions or updates

The *finite state automaton (FSA)* is the simplest state machine in terms of state representation, as the machine retains no information beyond the state's identifier. States are linked into a directed graph with edges labeled from a pre-defined set of inputs. Depending on the input received, the system follows an appropriately labeled edge from the current state, to reach its new state; the new state might also correspond to some action, e.g., making an utterance. While apparently simple, an FSA lacks versatility and maintainability, as the number of states and the complexity of the graph grows very quickly for any non-trivial task.

In the *Information State Update (ISU)* [159] model there is a single *information state* that contains arbitrary data structures, specifically defined for each particular DM. *Update rules* retrieve and update the information state in response to triggers, so that the effect of rule application is to update the information state, rather than to make a transition to another state. A *control strategy* dictates the order by which trigger conditions are tested and rules are applied. According to the *Plan Based* model, the designer sets the goals of the DM and a planner is employed to guide the system towards achieving them. In hierarchical plan based dialogue managers, such as RavenClaw [80], system goals are represented as a tree, where in order to achieve a goal, the system needs to achieve the children-goals. The dialogue state is again represented as a set of designer-defined structures. Such models have the advantage of separating domain dependent and domain independent parts of the dialogue management procedure, thus making the DM more tractable and easier to port to new domains, as the domain independent features could be reused. According to the *slot filling* and *information seeking* models, the DM's goal is to fill the blank slots in a form that contains all information needed in order to carry out a task. Very well suited for applications such as voice portals, the system will prompt the the user for slot values until it has what is needed in order to, for example, make a booking. Both models are typically represented using MDP, SMDP or POMDP. *Bayesian update of dialogue state (BUDS)* [90] is a method based on Loopy Belief Propagation [91] for updating the dialogue state in a POMDP slot filling DS and *Hidden Information State (HIS)* [51] further develops the state-space reduction idea in BUDS. In particular, HIS groups user goals with equal beliefs into *partitions* (equivalence classes) and maintains a single belief state for each partition.

5.4.2 Adaptive Dialogue Management

RL has been used for some time in dialogue management because if dialogue is modeled as a conversational game (or an optimisation problem) the RL paradigm fits the dialogue problem very well and the wealth of efficient and mature methods in the RL literature are immediately applicable. Although MDP is a very good model for dialogue management, it has several shortcomings when it comes to adaptive dialogue management: it cannot handle uncertainty well and cannot model complex actions very efficiently. Various models are aimed at overcoming these limitations, including *Hierarchical MDP (HMDP)*, *SMDP* and *POMDP*.

5.4.3 Learning dialogue strategies

Having in mind the different theories and methods of dialogue management, we propose an ADS development framework (or cognitive system / architecture) that implements the most prominent existing dialogue management theories. Our system is developed in C++ and Robot Operating System (ROS) [160] and provides a friendly and intuitive interface for ADS designers.

In order to allow the designer to use any of the dialogue models described in later sections, we need a universal data structure that each model can use. We therefore use Partially Observable Semi Markov Decision Processes (POSMDP) $\{S, T, O, \Omega, R, \mathbb{O}, \mu\}$, where S is the state space, T is the transition probability distribution, O is the set of observations (or system outputs), Ω is the observation probability distribution, $\mathbb{O}$ is the set of options which are composed of $< I, \pi, \beta >$ and μ is a policy over options $S \times \mathbb{O} \rightarrow [0, 1]$. $I \subset S$ is the initiation set, i.e. a set of states that trigger the option, π is the policy followed when the option is selected and $\beta : I \rightarrow [0, 1]$ is a termination condition. Complex system actions can be modelled as options and basic system actions a can be modelled as one-step options, if we set $\beta(s) = 1, \forall s \in S$ and if we only allow action a, i.e. $\pi(s, a) = 1, \forall s \in I$. This model can be reduced to POMDP, SMDP, HMDP, MDP and FSM. It can grow in a dynamic fashion, as the ADS transitions to new dialogue states, therefore reducing the amount of memory needed to store the whole dialogue state space. The dialogue state will be a set of user defined structs and automatically generated auxiliary variables, to cover the needs of all four dialogue models. We will also provide any additional necessary infrastructure (data structures, learning algorithms) for each dialogue model.

Moreover, the proposed framework provides mechanisms to define FSA, MDP, HMDP, SMDP and POMDP, which are automatically mapped internally to our POSMDP structure, with the appropriate parameter settings. More specifically, we have a POSMDP class, where classes representing the other Markov models will be inherited from, with some parameters fixed and non-adjustable (e.g. $\beta(s) = 1, \forall s \in S$, etc). The system's goals are incorporated into the reward function and the user's goals are represented as a tree, where necessary. The framework also provides the necessary transformation functionality to make sure it adheres to the selected dialogue model. We proceed to discuss what needs to be done in order to transform a POSMDP into each of the aforementioned models.

5.4.4 Transformation to Dialogue Models

A FSM is defined as a tuple $\{S, A, O, T, R\}$, where S is a set of (dialogue) states, A is a set of available actions, O is a set of possible outputs, T is a transition function that yields the next state of the system, depending on the current state and the action to be taken and R is a function that yields the output that the system generates when it transitions to a state. The only difference of an FSM with an MDP is that the output set of the MDP is the real numbers and that transition probabilities have been assigned [161]. That is, in MDPs, T is a probability distribution instead of a function. In order to represent FSM based dialogue systems, therefore, we will reduce our POSMDP structure in to an FSM as follows. We will use a POSMDP $\{S, T, O, \Omega, R, \mathbb{O}, \mu\}$, where the transition function T will be a subset of $S \times \mathbb{O} \times S$ instead of a mapping from $S \times \mathbb{O} \times S \rightarrow [0, 1]$ (i.e. transitions are deterministic) and Ω, in the same sense, will be a subset of $S \times \mathbb{O} \times O$ instead of a mapping from $S \times \mathbb{O} \times O \rightarrow [0, 1]$ (i.e. the output is deterministic). Options will also be single step options, to model basic actions only. The reward function R remains the same, but is defined over states and single-step options.

Plan based models use a tree to represent the user's goals. Similarly to the FSM approach, a planner is used to traverse the tree and attempt to achieve the goals. In order to achieve non-leaf goals the planner must first achieve their children goals, with goals at the leaves of the tree representing basic actions. We therefore use SMDP to model the goals of plan based DS and the policy μ to model the planner. The options set will be automatically derived from the goal tree and will comprise both single and multi step options (i.e. basic and complex actions). The dialogue state space will be a set of designer defined structs. In plan based systems, the goal tree does not necessarily dictate the order of goal achievement, as some user input may affect it. Our model can take this into account by designing the appropriate dialogue policy μ, which can yield the appropriate system actions according to the current dialogue state (which is affected by the system's actions as well as the user's input). If the designer wishes to build an adaptive plan based DS, we can "enable" the transition probabilities of the POSMDP, as we also can in the FSM case. Moreover, we can "enable" the observation probabilities to account for uncertainty in the interaction.

To model ISU DS, we use a SMDP (which is a POSMDP with "disabled" Ω) to represent the dialogue state space and is generated dynamically, depending on the interaction, as follows. At first there is only one node, the starting state. From the starting node we calculate a set of rules that are applicable and create a multi step option (a complex action). Each step in that option represents a rule, whose conditions are met at the dialogue state the option is available from. According to the control strategy (which in this case is the option's policy π) a rule is selected at each step and the dialogue state is updated. While such multi step options are being executed, the system is in the dialogue state update phase. When a rule is executed a new node is added to the SMDP to reflect the new dialogue state. The system again computes the set of applicable rules (rules whose conditions are met at the current state) and generates the corresponding multi step option. This is repeated until a final state is reached. Apart from options that represent rules, we have single step options to represent system actions (prompts etc). The dialogue policy μ will make sure that "rule" options are taken first, and when none is available, a "traditional" system action will be chosen. We thus make sure that after taking each action, the system will update the dialogue state according to the rules.

To model SF or IS DS, we need to reduce our POSMDP model to either POMDP, SMDP or MDP, according to the designer's needs (i.e. uncertainty modelling, complex actions, etc). Reducing to any of these models is done as described for the previously mentioned models, by "enabling" or "disabling" features of the POSMDP. The dialogue state is represented typically as a vector where some dimensions correspond to slot values and some contain auxiliary variables. In the next section we describe how we construct dialogue strategies in an online fashion, using complex or basic actions and how we are able to achieve the system's goals even when they change over time, or other events in the environment affect their achievement.

We can now present our novel development framework, able to represent the most widely used DM models, namely FSA-based, plan-based, SF/IF and ISU, using a "universal" data model that covers all of these approaches' information structures. We are now in the process of creating a POSMDP dialogue system, using this infrastructure, to assess the benefits and drawbacks of a POSMDP DM. This system combines the advantages of the four dialogue models, i.e. an ADS that can model uncertainty, support online complex action learning and multiple goal achievement, while also incorporating dialogue state space compression techniques similar to HIS/BUDS. The application is a household assistant, able to understand the user's intentions and manipulate objects. To achieve this, it combines basic “verbal”, “computational” and “physical” actions, such as say something, search a database, pick or place an object, into complex ones, such as report daily news or sort the objects according to colour. Since different users have different needs, the system will have to learn how to meet them by adapting the way complex tasks (e.g. breakfast) are achieved.

At the current state of development, we have used RobotOS [160] as our communication protocol, as this fits our current use cases. We expect to develop similar communication back-ends for other middleware as well in the near future. In fact, we are planning to exploit data abstraction concepts and techniques from RobotOS and other middleware and from Web Services architectures in order to abstract away from the specific *interaction capabilities* (voice, terminal, multi-modal) that a particular system offers. As a further step, we envision that a comprehensive theory of human-machine interaction will be developed, one that is able to circumscribe the possible calculations and (kinds of) information involved in deliberating about state transitions and actions. Such a theory will form the background for defining a conceptually and computationally simpler framework that is more amenable to both manual authoring and machine learning/adaptation.

5.5 Discussion

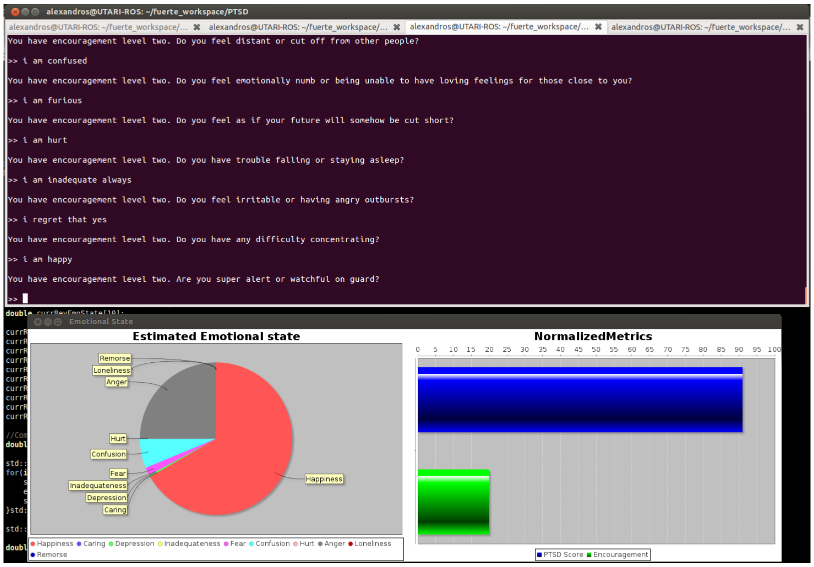
In this chapter, we have discussed possible applications of AWL and USE, mostly in an ALE context and discussed the impact such algorithms can have in DM. We also presented an ADS serving as a Museum Guide, which supports interchangeable learning modules and supports standard and state of the art learning algorithms. Last, we presented an ADS development framework, targeted mostly for non-experts in ADS development, to provide an intuitive and easy-to-use environment for developing systems for various applications.

In the following Chapter, an ADS targeted for Post Traumatic Stress Disorder (PTSD) assessment is described in detail. This system employs both AWL and USE algorithms, along with emotion recognition, in order to assess whether a person is suffering from PTSD, through natural dialogue conversation. The goal of the system is to make the user feel comfortable by not explicitly asking PTSD-related questions, but rather by implicitly extracting the necessary information (e.g. PTSD symptom identification) through seemingly casual conversation.

CHAPTER 6

Assessing Post Traumatic Stress Disorder

In this Chapter, we present an ADS targeted for Post Traumatic Stress Disorder (PTSD) assessment [148]. More specifically, the purpose of this ADS is to elicit information necessary to make an assessment of whether the user may be suffering from PTSD or not, through natural conversation with the user. It should be noted that his work has been conducted in close collaboration with mental health professionals (specifically, from UTA's Department of Psychology). This system is able to continually track the user's emotional state, based on a model proposed by [149]. When the user's emotional state "worsens", the system can adjust its *encouragement level*, providing information such as statistical data, to encourage the patient. The system (depicted in Figure 6.1) applies USE with a mandatory goal of eliciting PTSD-related information and an optional goal of keeping the user calm. More specifically, the present work describes our effort to meet the growing need for PTSD assessment systems, which we address by utilizing an ADS to handle Human - Computer Interaction. The ADS will provide a natural language interface (in spoken or written form) to users who potentially suffer from PTSD. Such systems have been proven to help motivate PTSD patients seek professional help [162]. Our system applies online reinforcement learning algorithms (HIAC) for adaptive dialogue management, paired with AWL and USE, which allow it to exhibit complicated behaviour and guide the course of the interaction in a way that achieves the system's goals. The information gathered is similar to the information contained in a self assessment test, and using this information we are able to identify symptoms relevant to PTSD and calculate a


alexandros@UTARI-ROS: ~/fuerte_workspace/PTSD
alexandros@UTARI-ROS: ~/fuerte_workspace/...
alexandros@UTARI-ROS: ~/fuerte_workspace/...
alexandros@UTARI-ROS: ~/fuerte_workspace/...
alexandros@UTARI-ROS: ~/fuerte_workspace/...
Emotional State
Estimated Emotional state
Remorse
Loneliness
Anger
Hurt
Confusion
Fear
Inadequateness
Depression
Caring
Happiness
● Happiness ● Caring ● Depression ● Inadequateness ● Fear ● Confusion ● Hurt ● Anger ● Loneliness ● Remorse
NormalizedMetrics
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
■ PTSD Score ■ Encouragement

```
You have encouragement level two. Do you feel distant or cut off from other people?

>> i am confused

You have encouragement level two. Do you feel emotionally numb or being unable to have loving feelings for those close to you?

>> i am furious

You have encouragement level two. Do you feel as if your future will somehow be cut short?

>> i am hurt

You have encouragement level two. Do you have trouble falling or staying asleep?

>> i am inadequate always

You have encouragement level two. Do you feel irritable or having angry outbursts?

>> i regret that yes

You have encouragement level two. Do you have any difficulty concentrating?

>> i am happy

You have encouragement level two. Are you super alert or watchful on guard?

>> 
```

Figure 6.1. Screenshot of the PTSD Assessment ADS.

PTSD score [163]. Last, this system will be available in two modes of operation, one that requires user registration and one that does not, to preserve anonymity. In the following section we introduce PTSD and briefly present some symptoms that our system attempts to identify.

6.1 Post Traumatic Stress Disorder

According to the National Institute of Mental Health (NIMH) [3], PTSD is defined as an anxiety disorder which, as the name suggests, may occur after a traumatic experience. According to statistical data provided by the Sidran Institute [164], about 5 percent of the population of the United States is currently diagnosed with PTSD, which is roughly 13 million people, costing the society approximately 42.3 billion dollars annually [3]. Moreover, 7 out of 10 people in the United States have been diagnosed to suffer from PTSD one or more times during their lives, according ton NIMH, with female patients being twice as many as male. There is a dire need, therefore, for efficient and affordable solutions to diagnosing and treating this disorder and

Table 6.1. A list of the most prominent PTSD symptoms [3]

Symptom	Expression
Re-experiencing	Frequent, upsetting thoughts Recurring nightmares Acting as if the stressful experience is happening again Strong feelings, physical reaction when reminded of the stressful event
Avoidance	Avoid talking or thinking about the stressful event Avoid visiting places or people related to the stressful event Not remembering important aspects of the stressful event Loss of interest, becoming distant Having suicidal thoughts Inability to express positive feelings
Hyperarousal	Having trouble falling asleep or staying asleep Being irritable and feeling in danger Not being able to concentrate for a long time Being easily startled

the proposed system has the necessary potential of becoming a widely used PTSD screening tool. There are, of course, other promising solutions to assessing PTSD, but before presenting them and discussing their benefits and weaknesses, we briefly discuss the most common symptoms related to PTSD. It should be noted here that there are many causes to PTSD, for example a person or one s/he loves goes through catastrophe, physical assault or experience war and that, in order to be diagnosed with PTSD, an individual does not need to experience all those symptoms, but does need to experience some of them (as described later in this Section) for at least one month. Table 6.1 summarizes the most prominent PTSD symptoms, as defined by the NIMH [3].

According to NIMH, the most prevalent symptoms of PTSD are to re-experience the stressful event that caused it, to avoid anything that reminds the patient of it and to exhibit hyperarousal. Re-experiencing the event may be expressed by frequent and upsetting thoughts about it, by recurring nightmares, by acting or feeling as if it were happening again, and by having strong feelings and physical reaction (such as sweat or increased heartbeat) when reminded of it. Symptoms indicating that the patient is avoiding reminders of the stressful event are expressed by making an effort to avoid talking or thinking about it, as well as avoiding visiting places and meeting people related to that event. Moreover, not remembering important aspects of the stressful experience also belongs in this category of symptoms, as does the loss of interest in activities, becoming distant, suicidal and unable to express positive feelings. Hyperarousal symptoms are expressed by having trouble falling asleep or staying asleep, by being irritable and feeling in danger, by not being able to concentrate for a long time and being easily startled.

The most widely accepted treatment of PTSD is through cognitive-behavioural therapy (CBT) or through medications (NIMH). CBT is conducted by a mental healthcare professional and may involve stress inoculation therapy, exposure therapy or cognitive restructuring. It should be noted here, that researchers have recently proposed to use Virtual Reality as an alternative to deliver exposure therapy [165], taking advantage of the wide acceptance of new technology. All these methods aim at changing the way users respond to their feelings, thoughts or to everyday situations, and can also help them deal with unhealthy situations that arise from their feelings or thoughts. Treatment using medications involves antipsychotics, antidepressants and medication that aids the users sleep or relax in general. Other treatment methods include acceptance and commitment therapy (ACT) and psychodynamic psychotherapy (PP). ACT aims at making users understand and accept their experience based on the idea that their suffering derives from their attempt to avoid their emotional pain, while PP focuses on how the users subconscious affects their behaviour. It should be noted that treatments are also available for cases of co-occurrence of PTSD and substance abuse. However, there is little evidence for these approaches, so other therapeutic options are still greatly needed. Fortunately, there are now some more effective treatments for PTSD that fall into the category of Cognitive-Behavioral Therapy, and include exposure therapies, stress inoculation training, and cognitive therapies [166]. The two most researched and efficacious of the exposure therapies include Prolonged Exposure (PE) and Cognitive Processing Therapy (CPT) [167, 168].

One of the first steps a person must take towards treating PTSD is admit he / she is experiencing a great deal of stress associated with a traumatic event. Apart from visiting a mental healthcare professional who can make a diagnosis, there are several other resources available, such as PTSD self-assessment tests. These can be taken either at a designated facility, or online. Such tests are usually anonymous, are typically based on the PTSD checklist (PCL) [163, 169], and provide an estimate of the whether a person suffers from the disorder or not. Other online resources include chat rooms, forums, blogs and portals, where users (or their friends and relatives) can find valuable information and support. [170] proposed an Adaptive Dialogue System (ADS), able to have natural conversation with PTSD users, generate output in real-time and provide assessment and feedback regarding their condition. In the present Chapter, we present in detail how we are able to assess and monitor PTSD through natural dialogue, as well as the complete implementation of this System. At this point, it should be clearly pointed out that the use of this System requires the involvement of a licensed mental health professional (e.g., clinical psychologist, psychiatrist, etc.) in order to immediately deal with any mental health crisis which may arise (e.g., suicidal thoughts/threats; threats to others; etc.). In the following Section, we provide a detailed description of other assessment methods.

There are a plethora of resources on the web, containing information and support to people suffering from PTSD, the most technologically advanced of which being

SimCoach, developed by the Institute for Creative Technologies at the University of Southern California [162, 171]. SimCoach currently targets military staff, suffering from PTSD or depression, with the goal of providing information and motivation to seek professional help. The system uses various embodied conversational agents (ECA), each with its own avatar, in order to model various personalities, such as a sergeant or a social worker. The main idea behind this project is that a user may feel more comfortable talking to a virtual human, anonymously, rather than to a real person. The U.S. government has also released more than 80 PTSD-related applications for mobile devices (NextGov.com), developed by various agencies. Video games have also been used to help treat PTSD (MilitaryMentalHealth.org; VentureBeat.com). Other PTSD assessment systems include on-line tests (Online PTSD Tests), where the user must answer a series of questions, typically derived from PCL. A score is then calculated, and the user receives feedback concerning his / her condition, as well as information on available resources.

While these methods are able to successfully assess users PTSD score [163], the systems behaviour is static throughout the interaction, without taking into account needs and preferences the current user may have. Another disadvantage of current approaches (such as online tests or forums) is the fact that users may feel stigmatized if they use them and refrain from visiting a mental health professional, whereas evidence exists [171] supporting the fact that if they interact with an artificial agent, in natural language, they are more likely to seek professional help. Our system mainly aims at implicitly detecting symptoms consistent with PTSD symptoms, without having the user directly answer a questionnaire. For this purpose, we opted for natural interaction through an ADS, whose potential as a digital medium has been demonstrated (Konstantopoulos 2010, Rizzo et al., 2012b) to not distract users (as new and exciting technology) but rather help convey useful information as well as derive useful conclusions (i.e. detect symptoms possibly related to PTSD). More specifically, Rizzo et al. (2011) and Buckwalter et al. [171], support the fact that interacting with virtual agents helps people feel more comfortable with their condition and about seeking help from health professionals. According to the authors, such systems help users overcome barriers such as availability, acceptability and accessibility (as defined by [172]). Konstantopoulos [154] provides evidence that ADS and ECA do not distract users but have a positive influence in achieving their goals. Based on such evidence, we use an ADS, the core of a virtual agent, to interact with users and not only assess PTSD but also monitor their progress over time (e.g. while they are undergoing therapy). Moreover, while other approaches focus on solely on assessing PTSD, we additionally monitor the users progress over time (both during a session and across sessions). It should be noted that our approach is different from a tele-health application in that our system automatically detects possible symptoms that may be related to PTSD and forwards this information to a mental health professional (provided the user consents). To summarize, we propose a real-time adaptive system, able to derive a PTSD score [163] through natural conversation with the user, in a manner similar to SimCoach, but different in that we continuously monitor the users emotional state and generate output in real-time. Our system can also be incorporated in a larger system, which may, for example serve as a companion. In the following Section, we introduce the concept of Adaptive Dialogue Systems (ADS), which we employ to handle the interaction between the user and the assessment system (which may be supervised by a licensed mental health professional, if necessary).

ADS are intelligent systems that offer a natural and intuitive interface when interacting with people. Such systems typically interact through spoken or written natural language, but are also capable of interacting through various other modalities, such as text, speech, video, gestures and others. This is what makes ADS ideal for our application, which is to automatically assess PTSD score through conversation with the user. Moreover, there is a lot of evidence supporting the fact that a trusting relationship between PTSD patients and their therapists is helpful to their treatment [173]. We therefore wish to investigate if convincing the user to believe he/she is interacting with an intelligent agent helps create a sense of trust between them, and subsequently makes the user more willing to share information. However, there are several challenges that need to be met before we can claim to have a System that appears intelligent, one of them being the Systems ability to lead or guide the conversation, and another being the ability to remember previous conversations with that specific user, as well as facts about him / her (such as family members, general preferences and special needs). To address the first challenge, we employ an algorithm that is able to influence the progress of conversation, thus helping achieve the Systems goals (which could be, for example, to get the user to elaborate on a specific PTSD symptom) in real time. For example, if the user keeps talking about topics irrelevant to PTSD, the algorithm could change the subject of the conversation and focus on a specific identified PTSD symptom by asking appropriate questions. To address the second challenge, we take the standard approach of creating a profile for each user, containing all the necessary information. Again, in addressing both of these aforementioned challenges, we need to again emphasize that a mental health professional will need to be integrated into the system in order to achieve maximum benefit. Details of how this will be worked are still being worked out.

The System described so far, as mentioned before, may be stand-alone or part of a larger system, acting as a companion, possibly installed on a robotic platform. To allow users who do not wish to register with the System, or to have private information stored as part of their user profile, we had initially considered having an online version of the System that would not require registration, similar to that of Rizzo et al [162]. However, because of the ethical issues discussed earlier, our System will need to be different because we continuously monitor the users emotional state, automatically generate output in natural language, and continuously adapt and improve its behaviour by learning from the interactions with registered users. In the following Section, we present some background knowledge concerning the ADS and how we are able to assess a users PTSD score [163]. We then proceed with a detailed System description, followed by our plans for evaluation.

6.2 Modelling the Interaction

To model the dialogue problem, we take the Information State Update (ISU) approach [50], which enables us to keep track of specific attributes, related to PTSD, as well as information regarding the progress of the current interaction. In more detail, according to the ISU approach, we have a dialogue state, which models anything of interest, such as identified symptoms, emotional state, name, age, gender, occupation and so on. The values of these variables are updated by a set of rules that are triggered by specific events, such as a new symptom being identified. The rules are typically set by the designer of the System but, in our case, we allow them to be updated during the interaction, in order to achieve adaptation to each user, as we describe later in this Section.

In order to adapt, the System needs to learn how to behave (i.e., how to respond to user utterances, taking into account information contained in the dialogue state). Before applying learning methods, however, we need to model the interaction as an optimization problem. To achieve this, we follow the Information Seeking [150] paradigm, according to which we have a set of possible user actions, a set of possible System actions, a set of slots representing pieces of information the System needs

Table 6.2. System and user actions, defined in our system

User Actions	System Actions
GREET	GREET
ANSWER(PCL)	INTRODUCE
ANSWER(Casual)	ASK-YES-NO
ASK(PTSD)	ASK(PTSD, Explicit)
ASK(Casual)	ASK(PTSD, Implicit)
END	ASK(Casual)
SAY-NOTHING	ANSWER(Casual)
	ENCOURAGE
	CONFIRM
	REPEAT
	GIVE-FEEDBACK
	END

from the user and a dialogue state, representing how the dialogue is progressing so far. Slots, in the PTSD assessment system case, may represent, for example, identified symptoms, their perceived severity, current emotional state of the user, and any other piece of information that is relevant to the application. The System will attempt to fill the slots, by asking the appropriate questions. For example, to fill a slot regarding the users current mood, it could ask: How are you feeling today? It should be noted that not all slots need to be filled before the System can make a reliable assessment.For this application, we define the user and System actions summarized in Table 6.2.

The users actions are derived by processing their input (spoken or written natural language) and classifying it in one of the seven available actions. In favour of simplicity, we classify all information-providing user utterances as ANSWER actions, which generally refer to answering a Systems question, such as: How are you feeling today? We only make one distinction, whether the answer refers to a casual question or to a PCL-related question. Similarly, ASK refers to the user asking either a casual or a PTSD-related question, for example: Where is the nearest treatment center? DESCRIBE refers to the user elaborating on something, such as a symptom or experience, and END refers to the user ending the interaction. SAY-NOTHING refers to absence of user input, for a predefined amount of time after the last System action. The Systems actions, on the other hand, are translated into natural language, as described in the following Section. GREET refers to greeting the user, INTRODUCE refers to the System introducing itself (as a virtual being), ASK-YES-NO refers to asking a question that requires a yes or no answer, such as: Do you have trouble sleeping? and ASK refers to asking casual questions, implicit or explicit PTSD-related questions (e.g., Why are you so angry?) ANSWER refers to the System answering casual questions posed by the user and ENCOURAGE refers to providing encouragement to the user, in an attempt to make him / her feel better. CONFIRM refers to an attempt to confirm something the user has previously said (e.g., "So you said you recently returned from the battlefield?). REPEAT refers to asking the user to repeat what he / she said last, GIVE-FEEDBACK refers to providing PTSD-related information, either regarding the user, general facts, treatment options, local available resources, etc, and END refers to the System ending the interaction. We define the dialogue state to contain the following information: d={ Answers_PCL, Symptoms, Encouragement, Score_PTSD, EmotionalState}, where refers to answers the user has given on (explicit or implicit) PCL questions; refers to identified PTSD symptoms; refers to the current estimate of the amount of encouragement the user needs; refers to the current estimate of the users PTSD score; and refers to the users current emotional state. As we mentioned before, the dialogue state is updated by a set of rules, for example if the user has provided an answer, we update and, if necessary, and . A dialogue policy, then, would dictate which of the available System actions the System should choose, taking into account the current dialogue state, in order to get the necessary information from the user and assess the severity of his /

her disorder. Being adaptive means that the System changes its dialogue policy (i.e., its behaviour) in response to changing circumstances or to current user preferences / special needs. This can be achieved by using standard real-time RL methods, such as SARSA [174], Q-Learning [141], Natural Actor Critic [92], which we used in our implementation, and others. Here, we want to emphasize once again that a trained licensed mental health professional is involved in this sequence at all times. If certain symptoms or answers are in the significantly abnormal level (previously determined), then they will be flagged and will alert the mental health professional about making immediate contact with the participant and possibly intervening.

As we have mentioned before, the goal of our system is to guide the conversation in a way that allows it to elicit from the user important information regarding PTSD. To achieve this, we need to be able to adapt the way we elicit this information, depending on each users personality, special needs and also depending on environmental changes. We also need to formally define the Systems goals, which may be characterized as mandatory or optional. In this case, we define one mandatory goal: elicit enough information to perform PTSD assessment (i.e., identify at least one symptom regarding re-experiencing a stressful event, three or more symptoms regarding avoidance, and two hyper-arousal related symptoms, as well as their intensity). It should also be noted that the user should be experiencing PTSD symptoms for at least one month (US Dept. of Veteran Affairs). Identifying the symptoms intensity can be achieved in a variety of ways but, in this version of the System, we only take into account the words the user chooses to use and assess the emotional state by comparing the emotional weight they convey. Optional goals may vary during the interaction, and can be defined as keeping the user calm and eliciting additional information about an event (for example, making the user elaborate on an experience), in an attempt to assess whether it can be classified as a PTSD symptom or not. In order to make

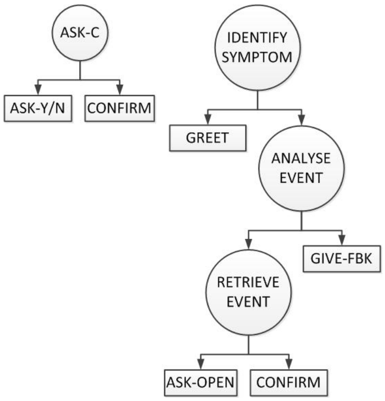


Figure 6.2. Example complex actions.

the System appear intelligent, besides real-time System behaviour adaptation, we use a method which allows the System to combine multiple simple actions into complex ones, thus making more complicated utterances. Such methods have been proven to have a positive effect on the interaction [175, 147] and we wish to investigate the fact that our method will be a step towards making the System appear intelligent. In the following Section, we provide more details on how this technique works, and how it affects the conversation.

To achieve more natural interactions, we use AWL, to allow the System to combine (basic) actions into complicated ones, thus learning how to achieve more difficult tasks, such as identify symptoms and also learning multiple ways to achieve a given task (for example, asking indirect versus direct questions, or providing feedback in various ways, according to what is more likely to be accepted by the user). This allows the System to exhibit complicated behaviour, and therefore the user perceives the System as intelligent and is possibly more willing to share useful information. More formally, instead of having System actions such as GREET or ASK-OPEN, we allow combinations of actions, represented as trees and forming complex actions.

A complex action, therefore, corresponds to a complicated system utterance (or to solving a task), by combining basic or other complex actions. Figure 6.2 depicts some example combinations of actions, where circles denote complex actions, and rectangles denote basic System actions. ASK-C corresponds to asking a yes/no question and then confirming the answer. Such an action can be useful, for example, in noisy environments. IDENTIFY SYMPTOM is composed of greeting the user and subsequently retrieving an event, analyzing it and providing appropriate feedback. ANALYSE EVENT comprises retrieving an event and providing appropriate feedback, according to the analysis and the current users profile. RETRIEVE EVENT is composed of asking an open-ended question and subsequently confirming it. In more detail, AWL assesses the performance of complex System actions, on previous interactions, and provides a prediction of successful outcome, if the System selects a complex action at a given time during the interaction. This allows learning how to combine basic and complex actions into more complicated actions, in order to solve specific tasks, as well as allowing learning different ways to achieve the same task. The action predictions may change during the interaction, and the System will adapt by selecting a different complex action, in response to the users utterance. The chosen complex action is subsequently passed on to the appropriate output generation modules, which can then create more intricate sentences. Another benefit of using complex actions is the ability to design more abstract behaviours, as well as to transfer knowledge of how to achieve certain tasks from one system to another.

In order to be able to guide the conversation with the user, and elicit information regarding PTSD, we employ USE, which provides us with the estimated effect a System action may have on the user and the environment. Using these estimates, we (including the licensed mental health professional) can make informed decisions on how to proceed with the dialogue, taking into account the interaction so far, as well as other information we already know about the user (e.g., through his / her profile). Having presented the methods we employ to handle the interaction at a low, decision-making level, we now proceed to discuss the overall architecture of the System, and present other modules we employed or developed, necessary to process the users input and the Systems output.

6.3 PTSD Assessment ADS Overview

In this Chapter, we have presented an ADS able to interact with PTSD users through natural language dialogue, and provide a basic assessment of their PTSD score [163]. To achieve this, we opted for a modular system design, where each module is responsible for a specific task, such as deciding how to respond, generating output in natural language or keeping track of the users emotional state. In the rest of this Section, we present the architecture of our System, describe each module in detail, discuss how we are able to assess PTSD, as well as how we generate feedback to the user. Lastly, we present some example interactions between our System and a real user (not a PTSD patient). In this Section, we describe in detail the architecture of our System (depicted in Figure 2), where ASR stands for Automatic Speech Recognition, NLU corresponds to Natural Language Understanding, Emotion Recognition (ER) refers to emotional state estimation, NLG corresponds to Natural Language Generation, and TTS stands for to Text To Speech. The input modules are responsible for parsing and processing the users input, which may be in spoken or written natural language. In the first case, speech is intercepted by the ASR module, which converts it into text. This is then passed on to the NLU component, responsible for understanding the meaning of the users utterance. This information is subsequently passed on to the ER module, which provides an estimate of the users current emotional state.

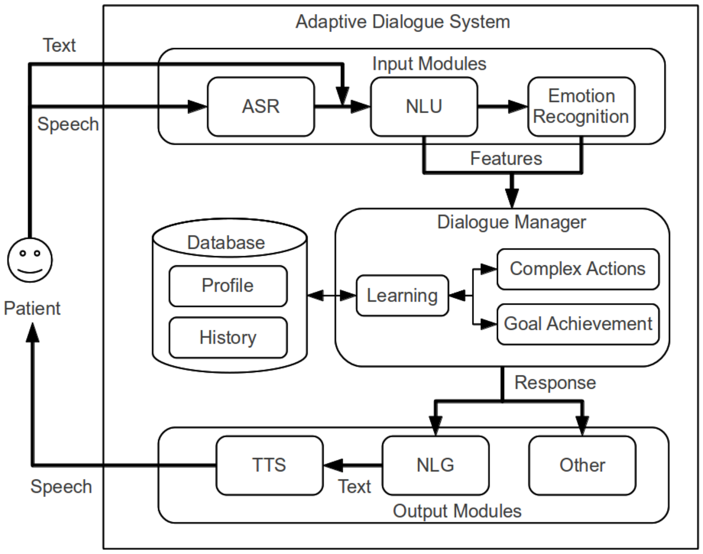


Figure 6.3. The architecture of our Adaptive Dialogue System, for assessing PTSD score. Spoken input is captured by the Automatic Speech Recognition module, which transforms it in textual form, which is then intercepted by the Natural Language Understanding and Emotion Recognition modules, which extract useful information. This information is passed on to the Dialogue Manager, which decides how the system should respond, and this response is translated into spoken output by the Natural Language Generation and Text To Speech modules.

Output from both the NLU and ER modules is then processed by the dialogue manager, which is responsible for deciding how to respond to the users utterance, as described in the previous Section. All learning and adaptation modules are implemented within the dialogue manager, which also interacts with a database containing information about the current user and previous sessions. It is at this level that the mental health professional can intervene if any suggestion of harm by the user to himself/herself. The dialogue manager produces a System action, which is received by the NLG module which, in turn, generates the Systems output in natural language, also using information contained in the dialogue state. This output is then passed on to the TTS module, which generates spoken output. It should be noted that a dialogue

System may generate output in other forms, such as graphs, playing videos, moving a robots actuators, etc, which will be included in future versions of our System.

In the following Sections, we present in detail the most important modules that comprise our PTSD assessment system, namely Natural Language Understanding, Emotion Recognition, Goal Achievement, Natural Language Generation and the User Profile System. The NLU module is responsible for parsing the users utterance, derived from the ASR component, for which we use a well-known open-source recognizer, named pocket sphinx [152]. In our NLU implementation, we parse the user's utterance, aiming to retrieve various kinds of information (depending on the state of the dialogue), such as emotional state, assessing user's response (positive or negative), or identifying symptoms. To estimate the user's emotional state, we mainly look for certain keywords indicating an emotion and its intensity. To assess the user's response to the System's direct or indirect question, we attempt to match the response against positive, neutral or negative patterns, such as Yes, very often or I don't know, that will also provide us with an indication of a Likert-scale-like response, that we can use in the PTSD score calculation. To identify a symptom, we again look for certain patterns, and we make the strong assumption that the user is still referring to the same topic and is not trying to divert / avoid the subject or talk about another symptom or experience.

Because we are dealing with potential patients, suffering from a psychosocial disorder, it is important to reliably track their emotional state throughout their interaction with our System. To achieve this, we model the users emotional state using the model proposed by Chuang and colleagues [176], and track their emotional state by looking for relevant keywords or phrases, defined by Drummond [177]. When we detect a deteriorating emotional state, we raise the encouragement level the System provides, in an attempt to keep the user in a calm state. If this does not seem to work, the System will make only casual talk for some time and, if even this does not work, the System will raise an alert to the administrator (i.e., the licensed mental health professional) for immediate intervention. According to the model we used, we have the following emotions the user can express: happiness, caring, depression, inadequateness, fear, confusion, hurt, anger, loneliness, remorse. Before we can make an update on the users emotional state, according to that model, we need to assess the emotions expressed by the most recent user utterance. To achieve this, we use an emotion vocabulary [177], according to which several words are tied to expressing light, medium or strong emotions, and by also taking into account words or phrases, such as not really, extremely, a little bit, etc., that can modify the strength of each emotion. We are able to make an estimate of the emotions conveyed in the users utterance. It should be noted that each emotion has a pre-defined positive or negative effect, which we also take into account before estimating the overall change of the emotional state. The NLG module is responsible for translating the Systems response into natural language, in spoken and textual form. We used simpleNLG [178] for this purpose, which is a publicly available, open-source framework for NLG. The Systems response can be categorized into information requests, feedback, encouragement and casual talk. Information requests refer to attempts to elicit PTSD-related information from the user, and can vary from straightforward questions (similar to PCL questions) to utterances which implicitly try to get the user to admit or reject something. Feedback refers to providing a verdict, summarizing the results the System calculated, based on the interaction with the user and PTSD assessment methods, as well as any information that could be useful to the user, such as locally available resources. Encouragement refers to providing statistical facts or other types of information, aiming to make the user feel better. Casual talk refers to output with no apparent purpose, other than to relax the user and subsequently attempt to elicit information.

The User Profile System is responsible for storing and maintaining useful information about each user. To promote anonymity, we allow the user to set a nickname of his / her choice, to be used by the System during the interaction. Other information stored includes data about the user's emotional state, as it evolved during each session, data about the user's PTSD score after each session and symptoms identified and/or discussed at each session. Location information is also stored, when available, to provide better information about PTSD-related resources that might be available in the users vicinity. Using this information, the System is able to form a sense of trust with the user, by creating the illusion that it remembers facts mentioned by the user, thus appearing more intelligent and not just a question-answering dialogue system.

In this Section, we describe the methods we use to calculate a PTSD score, based on PCL (civilian or military). The process we follow is standard, and comprises identifying important events that lead to PTSD, calculating a score using PCL guidelines [163], and generating appropriate feedback. More specifically, we need to identify a minimum number of symptoms from each category, during the interaction with the user. As we have mentioned in previous Sections, when identifying an event, we assume the user's response remains on topic (i.e., he / she does not attempt to avoid or change the subject). While this is not an unreasonable assumption to make, we plan to dismiss it in future versions of the System, by using more sophisticated NLU methods. When the ADS, therefore, directly or indirectly asks the user to elaborate on a certain answer (or on a previously identified significant event), we search for emotional keywords and assess the emotion conveyed by the users description. When a certain threshold is surpassed (for example, regarding fear or anxiety), we consider it a significant event. It should be noted here that these thresholds are defined manually and some trial and error may be necessary, before deploying the System. To calculate the users' PTSD score, we assess their responses to System questions, and classify them into Likert scale scores: Not at all (1), A little bit (2), Moderately (3), Quite a bit (4), and Extremely (5). Combined with significant events we have identified, we are subsequently able to calculate an overall PTSD score, following suggestions found in the literature [163, 164, 3]. An estimate of this score is dynamically calculated and, as the System gathers more information, the estimate reflects the true PTSD score. This estimate, in addition to information about the user's current emotional state, is essential for deciding whether to provide encouragement and which arguments to use. If the System detects a PTSD score greater than a cut-off value of 45 [163], it will provide feedback to the user, suggesting available PTSD resources (i.e., online resources, telephone numbers, health professionals and institutions near the users location, etc.). The System will then forward the conversation to a mental health provider, along with any available historical data regarding the particular user (e.g., previous PTSD scores, previously identified symptoms, and other relevant data). It should be noted here that the mental health professional will have the opportunity to change the cut-off value, depending on each users special needs.

As mentioned in the previous Section, the System uses a variety of information in order to generate appropriate, meaningful and helpful feedback. One general rule that we follow when generating feedback is making it clear that the System cannot really understand what the users have been through, but it is here to provide assistance by allowing them the chance to share their traumatic experience with an artificial human agent. Besides the core conversation, there are two types of feedback the System provides: information related to PTSD diagnosis and treatment, provided at the end of each session; and encouraging facts or tips that will help the user and keep him / her in a relatively calm state throughout the interaction. In order to be able to provide appropriate feedback (e.g., we state that the user possibly exhibits symptoms consistent with PTSD symptoms), we provide useful information (available resources, taking into account the users location if available) and present it in an appropriate way (for example we may use different wording according to age). Even if we do not detect PTSD, we provide some information to the user, pointing to helpful resources, because maybe we have been unable to detect PTSD or the user may have another disorder that our System cannot detect. In any case, since the user took the time and effort to interact with a PTSD assessment tool, we feel obligated to provide assistance. Again, our mental health professional is included at each stage of this process. As noted earlier, our System is able to calculate the amount of encouragement the user needs in order to remain calm during the interaction, and adjust its output accordingly. More specifically, we have five levels of encouragement, ranging from no encouragement to only providing encouragement (and not trying to elicit information or talk about other subjects). In order to estimate the amount of encouragement the user needs in order to become calm, we classified the recognized emotions into a scale ranging from 0 to 1, with 0 representing an emotion far from calm and 1 representing an emotional state very close to calm. We also need the last estimate of the user's emotional state, to assess its progress (whether it is decreasing or increasing). The level of encouragement is calculated as follows:

$$encouragement = \sum_{i=10}^{10} w_i e_i \tag{6.1}$$

where the weights represent the negative or positive effect an expressed emotion has on the user's overall emotional state.

We now provide some example interactions between our System and a human user (not a real PTSD user). After each turn, the System updates its estimates of the user's emotional state, PTSD score and encouragement level, as shown below.

User: hello [Emotional State: [Happiness 0.5, Caring 0.5, Depression 0.5, Inadequateness 0.5, Fear 0.5, Confusion 0.5, Hurt 0.5, Anger 0.5, Loneliness 0.5, Remorse 0.5]PTSD Score: 0, Encouragement: 0]

System: Hi, how was your day so far?

U: just the usual, nothing much [Emotional State: [Happiness 0.25, Caring 0.1, Depression 0.15, Inadequateness 0.1, Fear 0.1, Confusion 0.1, Hurt 0.1, Anger 0.1, Loneliness 0.1, Remorse 0.1] PTSD Score: 0, Encouragement: 0] **S**: I see. Was the weather nice in Dallas today?

U: yeah, it was kind of sunny [Emotional State: [Happiness 0.25, Caring 0.1, Depression 0.05, Inadequateness 0.05, Fear 0.05, Confusion 0.01, Hurt 0.01, Anger 0.01, Loneliness 0.1, Remorse 0.05] PTSD Score: 0, Encouragement: 0] **S**: Im happy to hear that. So is there something that has been troubling you today, or did you just want to chat?

U: I dont know, I dont feel very good I guess [Emotional State: [Happiness 0.05, Caring 0.05, Depression 0.15, Inadequateness 0.15, Fear 0.01, Confusion 0.05, Hurt 0.15, Anger 0.15, Loneliness 0.15, Remorse 0.15] PTSD Score: 0, Encouragement: 1] **S**: I see. Its good that you want to talk about it. Have you been having enough sleep lately?

U: Well, sometimes i have nightmares. It always makes me so angry that I can't sleep well. [Emotional State: [Happiness 0.05, Caring 0.05, Depression 0.25, Inadequateness 0.05, Fear 0.01, Confusion 0.25, Hurt 0.15, Anger 0.5, Loneliness 0.05, Remorse 0.05] PTSD Score: 1, Encouragement: 2] **S**: I can understand your frustration, its really bad when you cant sleep well, but I know there are a few things that can help you sleep. Are your dreams always about something specific?

According to [152], the pocketSphinx ASR system had a reported word error rate of 9.73%, meaning that 90.27% of the time the system was able to correctly recognize the users spoken input. It should be noted here, that this greatly depends on the language model used, which defines words and phrases that typically occur in a given domain, each with a probability of occurrence. For the purposes of our system we have developed, in collaboration with medical health professionals and the literature, a custom language model which we hope captures most key phrases and words (i.e. PTSD-related as well as casual talk) that will occur during interactions with our System. Regarding emotion recognition, the model proposed by [176] was reportedly able to correctly recognize emotion 81.49% of the time. We pair this system with an emotional vocabulary [177] and thus expect to at least maintain that level of accuracy. While this does not directly affect the calculation of the PTSD score, it does affect the identification of symptoms and we therefore include it in our reliability analysis. Last, for calculating PTSD score, as mentioned before, we use the PCL questionnaire [163]. Based on these facts, we expect our Systems reliability to be at least 0.98199. At each dialogue turn we only need to recognize some keywords. The system may also attempt to recover if recognition confidence is low.

6.4 Usability Inspection and Evaluation

Here we present the results of the usability inspection we performed, in order to evaluate the interface and usefulness of our system. We opted for heuristic usability inspection, using some of the heuristics proposed in [179, 180, 181, 182], which are relevant to our application. Our assessment showed that our system satisfactorily addresses these issues, as summarized in Table 6.3, below.

In the future we also plan to evaluate the Systems usability and performance with human users, suffering and not suffering from PTSD. In the next Section we describe in detail our plans for evaluation.

Table 6.3. Usability inspection of our system

Usability Heuristics	System
Visibility of system status	The user reads and / or listens to the systems output. The health professional views metrics, such as estimated emotional state and identified symptoms.
Match between system and real world	The system uses natural language as an interface, including common terms and only uses technical terms where necessary (e.g. when providing PTSD-related information).
User control and freedom	The user is able take the initiative at any point during the interaction.
Consistency and standards	The learning algorithms used guarantee consistency in the systems behaviour.
Provide clearly marked exits	The user is always able to type or tell the system to quit, navigate the systems menu and select quit, or can click an ‘X’ button.
Recognition rather than recall	The user does not need to remember how the system works.
Flexibility and efficiency of use	The user interacts with the system through natural language and may take initiative at any point in the dialogue.
Speak the users language	The language used by the system is plain enough to convey useful information and does not contain difficult to understand terms.
Minimize users memory load	The user does not need to remember how the system works.
Provide feedback	The system constantly provides feedback to the user and the health professional.

6.5 Evaluation

In this Section, we present our planned experimental setup. We will initially conduct Wizard of Oz experiments with real users, not suffering from PTSD. More specifically, this type of experiments involve having a human user (operator) playing the role of the System, and interacting with other users, who believe they are interacting with the System. This allows us to collect dialogue interaction data, which we can later use to train our System and learn a dialogue policy (system behaviour) which is able to correctly assess PTSD score (Weathers et al., 1993). It will also allow us to collect control data, i.e. responses from non-PTSD users, which we can later use to compare against potential users. Each user will be given a story containing one or more stressful experiences, and will be asked to interact with the system. Each story will be correlated with a PTSD score that the System (operated by a human user in this experiment) will need to correctly estimate. The System operator (a mental health professional) will have clinical guidelines on how to respond, taking into account his / her perception of the users emotional state. We will then use the dialogue corpus collected to develop user simulators that will, in turn, be used to train our System. In order to evaluate the performance of our System, we will compare its behaviour against a handcrafted dialogue policy. This policy will have three phases: a casual, introductory phase, where the System will ask questions including known facts about the user (such as age / gender) and perhaps asking for more (for example if the user likes where he / she lives); a PTSD information phase, where the System will implicitly or explicitly request PTSD-related information; and another casual phase, where the System will attempt to end the conversation in a way that invites the user to interact with the System in the future while also providing feedback concerning PTSD. The first phase will end after a pre-defined maximum number of dialogue turns, while the second phase may alternate while the system tries to get the user to relax (e.g., by providing encouragement). The final transition from second to third phase occurs when the System has enough information, when the user becomes too frustrated, or when the user decides to end the interaction. Since most ADS which apply RL methods are evaluated against user simulators, we will subsequently train our System with agenda-based user simulators [123], specifically designed for this problem, following a model of a PTSD user, with various personalities, experiences and PTSD score that needs to be correctly estimated. This design will be carefully tailored by the mental health professional.

The next phase of the evaluation will be to: test both systems on PTSD-suffering users; assess them; and compare their performance with other methods, such as PTSD self-assessment tests either online or not. Two groups of users will be formed randomly, one of which will use our System and the other will use other methods. Questionnaires will be completed by the users at the end of the experiment to assess their interaction experience.

6.6 Discussion

In this Chapter, an ADS was presented, which is able to interact with users suffering from PTSD, using natural language, and by guiding the conversation in an appropriate way. It is able to retrieve information from the users, which is necessary to assess PTSD. The purpose of our System is to reach out to users who are unwilling to talk to a trained mental health professional in person about their condition, but may be more open to interacting with a virtual artificial agent. The System will attempt to assess users PTSD score [163], provide them with useful information regarding PTSD and encourage them to take the next step towards treatment. Another benefit is the generation of large amounts of data that therapists may search and analyze. As technology advances, we have more survivors of war, natural disasters and other stressful experiences, thus an increased probability for PTSD in any given society. On the other hand, we have an array of tools available to detect abnormal behaviors and help people in need, such as our ADS PTSD Assessment System. Furthermore, the proposed System can easily be adapted to target other populations, such as people suffering from depression. In the future, we plan to enhance the speech and emotion recognition accuracy by using more modalities, such as audio, video or depth and apply Discriminant Laplacian Embedding [183] and audio-visual speech recognition. Finally, we plan to have an online version of the System available for testing.

CHAPTER 7

Discussion

Throughout this thesis, our contributions to the fields of Assistive Living Environments and Adaptive Dialogue Systems have been presented. While we are still far from self-aware artificial agents, the proposed algorithms are a step closer to creating an agent able to make casual conversation with people. In this Chapter, we present our concluding remarks, we briefly discuss some of the still open problems we chose not to address, we present our immediate plans for future work and, last, we discuss the potential impact of our contributions on the society.

7.1 Open Problems

As technology advances, new research fields are born, which frequently are amalgam of various sub-fields. It is unrealistic, therefore, to claim that we can address all the open problems in a research field as broad as ADS in a single PhD dissertation. In this section we present issues (existing or discovered along the way) that we opted not to address. One of the most critical, in our opinion, is that an ADS should be able to adapt within the duration of an episode, i.e. before the current interaction terminates. While there is related work in the literature, such efforts are very limited in scope and applicability. It might be useful to examine methods from other fields, such as tracking techniques, specifically designed to adapt to changing targets or changing target locations. Something related to this problem is rapid learning of good dialogue policies, in an online fashion.

While significant advances have been made during the past few years in action-state space approximation for ADS, we still need radically different techniques, that completely alleviate this problem rather than try to address it, in order to create truly intelligent ADS. Realistic user simulators are also of great importance, if we are to train robust and complex behaving ADS. Such simulators should model errors, error recovery strategies (from the user's point of view) and have variation in order to avoid overfitting. Another very important open problem is that of DS evaluation. A significant amount of work [99, 157, 126, 184, 130] has been done towards this direction, but it remains open as if is really difficult to assess user's satisfaction, especially while they are interacting with the system rather than post-evaluation studies (questionnaires etc.).

7.2 Future Directions

While there are a lot of open problems regarding DM, such as intra-episode adaptation, initiative taking, interruption handling and others, they can be tackled to a good extend by a change in paradigm, whose need has been suggested in the literature [62]. In this Section, we present our thoughts and ideas concerning the future of ADS, focusing on adaptive DM as the core technology. It is our belief that, before aiming for fully autonomous ADS, capable of lifelong learning, it is necessary to go through a semi-autonomous phase, where the user and the ADS will collaboratively accomplish tedious, dangerous and other tasks and learn from each other. With such systems, we can get the society interested and allow ADS to be widely adopted. As more people are drawn to ADS, both users and researchers, new problems and creative solutions will arise. We believe the first step towards semi-automation is to focus on methods where the user can teach the ADS (or robot), to some extend, how to perform a task. The paradigm should not be that the system is brilliant and can accomplish everything, but rather that it collaborates with the user and they both learn how to solve a task. The human should be kept in the task-accomplishing loop, and we should spend some time developing methods that take this into account. Up to date, this has been done mainly due to necessity, but we argue, here, that such systems should be deliberately designed this way. Following application, acceptance and success, we can shift focus to more automated methods, and we will have much more experience and intuition to do so. We envision a robot, controlled by a DS, able to navigate through obstacles and manipulate objects in its environment, that will work collaboratively with (i.e. will not replace) the user, to achieve a complex, dangerous or tedious task, while also being able to adjust or allocate its resources accordingly, in response to changing circumstances. Tractable methods should be designed, not only regarding the problem size (i.e. the system's ability to reason about many things) but also regarding length of interaction, as a lifelong virtual companion. Data should be stored (and possibly compressed) in appropriate formats according to frequency of use. In the rest of this Section, we focus on certain ADS aspects and attempt to envision possible directions of research that will lead to the next generation of ADS. We view each aspect as capabilities that can be built on top of a modular, adaptive DM.

7.2.1 Dialogue Management

We believe that dialogue managers should be decoupled from specific applications and their design should be modular, aiming to re-use modules across managers and platforms. The modules that compose a dialogue manager should have very well defined interfaces, to promote re-usability and facilitate seamless communication between them and other components. A module could, for example, handle user input from a specific modality, or model various behaviours that a certain artificial personality is expected to exhibit. Designers would then select the appropriate modules that fit their needs and save time and effort when building a new ADS. Modules would not even need to run on the same system, but generic DM components or those which require heavy computational power, could be implemented on the cloud or on secure remote servers.

Another, more practical aspect of DM (and DS in general) is the lack of a common infrastructure, accepted by the majority of the community, where researchers can develop and evaluate new methods, share knowledge and code. Despite various great efforts, such as [80], possibly due to hardware or operating systems constraints and preferences, they have not been fully adopted. We believe that a framework on such a shared infrastructure would allow other communities to easily develop ADS for their research, thus not only helping disseminate this technology and helping address existing issues, but also helping identify issues that our community was unable to predict. Such an infrastructure would moreover provide researchers with the ability to directly compare their novel methods with the current state of the art, or with well accepted benchmark problems (challenges), implemented using the common infrastructure.

7.2.2 Automatic Feedback Estimation

Since the goal of ADS is to interact successfully with humans, ideally the system should be able to pick up most of the communication signals produced by its interlocutors. Following the human paradigm in the semi-automation framework, an intelligent DS should be able learn from its instructor. To achieve this, we should have mechanisms to quantify positive and negative feedback, that the learning algorithms will utilise and learn how to react to similar situations. To reduce exploration while learning, we could develop acceptable hand-crafted policies (behaviours) to be used as starting points. At first we can have yes / no feedback, either from verbal cues or from gestures, facial expression, estimated user emotional state and body language. In the realm of semi-automation, we can define special training protocols, where instructors will use a predefined set of gestures or words in order to train their system. This process can then mature and become more intuitive to the human nature. In short, we need to teach intelligent systems how to learn, and then let them learn. Receiving feedback from common knowledge as well as from humour or sarcasm is also very important, but, it is our opinion that this is something to be looked at after ADS have mastered basic social skills. An important issue, raised by ADS researchers as well as by other research communities, is identifying whether it is the system that adapted to the user or vice versa, and if they reach some equilibrium, how can we quantify the amount of adaptation of each side.

Again looking at the human learning paradigm, humans are taught how to behave and identify what is good or bad. They associate certain words such as “No” with negative rewards, due to “pre-defined behaviours”, i.e. instincts, which make them afraid to yelling, or feel good in the opposite case. Another way humans learn is by observing the consequences of their actions, i.e. they are deprived of a desired (sometimes essential) resource, or they simply do not achieve their goals. Based on this paradigm, we can design methods and systems that employ statistical techniques to learn how to estimate feedback, beginning from some basic ground truth and building on it. One way to jump-start automatic feedback estimation is to begin with well known and unquestionably positive, neutral and negative human behaviours, and form this ground truth.

7.2.3 Complex Action Learning

Taking into account recent progress in this sub-field of ADS, we believe the next step is to develop robust mechanisms for learning from demonstration. The system must be able to map user actions into internal structures representing complex system actions. For example, when the user demonstrates how to use a tool to open a can, the system must understand which basic actions the user is performing, that are generalisable to the can opening problem. This can also apply to *Verbal* actions, where the user, for example, demonstrates how the robot should behave in a given situation (which questions to ask, how to ask them, which gestures to use, etc.). The system moreover needs optimisation mechanisms that will select the optimal way of achieving a task, if more than one ways exist. In our framework of semi-automation, the user should use a well defined and structured language or set of gestures to demonstrate the solution to a task. Definition of the task should be implicitly defined by this procedure. If a mapping from specific user directions to system actions does not exist, the system should first attempt to construct complex actions by combining available actions and if that fails as well, it should ask the user for simpler directions.

Knowledge should be transferable across tasks, as has also been suggested [93], and the system should be able to infer how to achieve similar tasks to the ones it already knows. Two other aspects of CAL, besides synchronization, are learning which actions should be executed in parallel, as well as adaptively managing the system's resources, when necessary. If, for example, resources are limited either due to design restrictions or due to unexpected events, the system should be able to prioritize actions and modalities to be used for input processing or for output. Such methods would have a great impact not only on the perceived intelligence of the system but on scalability as well.

7.2.4 User and System goal achievement

To achieve seamless and natural interaction, it is important to have an efficient and flexible model to represent user and system goals, as well as support mechanisms, able to detect goal dependencies, conflicts and assign priorities. Goals should be structured and ordered in a hierarchical or in a parallel fashion, depending on resources and on the nature of the goals. Moreover, the system should be able to predict the effects of a goal's achievement on the progress or feasibility of other goals. ADS should also incorporate techniques for goal learning, meaning automatically generating goals per user request, whether explicit or implicit. Papangelis et al. [2], proposed a method for modeling mandatory and optional system goals, that can guide the policy learning algorithm to specific dialogue policies which have high probability of achieving the system's goals. This approach also supports independent or conflicting goals, making sure that mandatory goals are reached, while optional goals are reached when possible. Such a method can complement state of the art goal modelling and inference methods found in the literature, and provide a DM with the aforementioned desired capabilities.

Moreover, goals should be transferable across modules, systems and platforms. They should be expressed in a common language that other intelligent systems can understand and process requests. For example, a museum can have several virtual agents or robots controlled by ADS, that can outsource computationally heavy tasks to a central museum server. Another example can be a set of lightweight (in terms of computational power and storage) agents with various personalities, who share a common computational resource.

7.2.5 Socially Intelligent Systems

One of the most important social skills an ADS should have, is to understand when and which people are talking to it rather than to each other (or on their phone). Another equally important skill, when it comes to robot DS is maintaining eye contact with a person or a group, or being able to identify that person again and remember their interaction, at least for a short period of time (similar to human short-term memory). Related methods exist in the literature of other fields (maintaining eye-contact, face recognition, etc.), but need to be seamlessly integrated in the context of an ADS. We also need to take into account the number of people the system is simultaneously interacting with and the social dynamics thereof. For example, it may be enough to only interact with the "leader" of the group or it may be necessary to interact with everyone, as a teacher does in a classroom. Such dynamics, of course, do not only apply to eye-contact or face recognition, but to all aspects of interaction. Such skills would greatly compliment efforts mentioned in previous Chapters, which include handling back-channels and gestures to provide information and control the dialogue as well as interpreting the users' back-channel signals and gestures (e.g. understand when a group of people becomes bored or nervous). All of these social skills can be crucial when training specialized ADS, such as question-answering, negotiation or other tactical systems.

An ADS or RDS should moreover know how to begin a conversation, or approach a person or group of people, respectively, as well as how to end the interaction. Understanding humour, sarcasm, distinguishing metaphors and fixed expressions from linguistic or semantic errors that are often left uncorrected in spoken language, are also important traits of a social ADS, the next step being a system able to express humour, irony and other complicated human communication intricacies when it is socially appropriate. Depending on the application, general social rules or behaviour can be expressed using mandatory and optional goals, that can change while the system operates, depending on the dynamics of the environment. Coupled with online complex action learning methods, which can provide the system with multiple ways of achieving a task (e.g. multiple ways to greet culturally different groups), such a system would have great potential in being perceived as socially intelligent.

7.2.6 User Simulators and Evaluation

User simulators have been a subject of debate regarding ADS training and evaluation. Creating realistic simulators is of paramount importance, as this is what the ADS's learning modules will adapt to. Given our current understanding of human cognition, we can neither perfectly simulate it at the signal level nor perfectly model it at the intention level. What would, however, be interesting to investigate is ways to transfer the strengths of each approach across to the other. Application-specific simulators are also necessary, for example, we need complex behaving simulators in order to train negotiation systems, where the simulator would attempt to deceive the system, provide false arguments, deliberately not reveal information or mislead the system against common sense.

Evaluation of ADS is also an open problem, despite the great efforts by the community, that have produced remarkable results. This has mainly to do with human nature, as we wish to measure user satisfaction, which is really hard to quantify, as each person is different, views himself and the system from a different point of view, and has different expectations, than any other person. The best way to evaluate ADS, in our opinion, is by measuring their acceptance by society. We should, therefore, strive to penetrate various markets, such as gaming, healthcare, manufacturing, etc., using open source systems or ADS development frameworks, that are easy to use by non-experts. As we have mentioned before, a major challenge regarding evaluation of adaptive systems is to detect whether the users have adapted to the system or the system to the users. Moreover, if both have adapted to each other, we should develop techniques that measure the degree of each side's adaptation. In an ADS that is in the everyday life of a person, it may be more likely that the person will learn how to use the it, viewing it as equipment, rather than the ADS adapting to the person. According to [80], comparing two dialogue systems is like comparing two programming languages. Each has advantages and drawbacks on specific tasks but we cannot exactly claim that one is better than the other, in a specific task, since the task can be re-modelled in a way that both systems perform equally. It is hard, therefore, to create benchmark ADS that other systems will compare to, as we would need a very wide range of application-specific systems.

A standardized evaluation methodology could include the following tasks, after the users have interacted with the system. First, the designers could set up a task for the ADS and evaluate the result. If, for example, the purpose of the ADS is to book tickets, we can measure the accuracy of the bookings versus time (or number of interactions) needed to get the relevant data from the human. As this scenario probably involves a trade-off, Receiver Operating Characteristic curves should be useful. As also mentioned earlier, an external observer may be present to evaluate the interaction. The designers should ask the users how much they enjoyed the interaction, breaking satisfaction down into several features. Likert scales would be an appropriate way of obtaining a numerical estimate from the users, as they have proven work very well for this purpose. A structured interview, between the observer and the user would be useful, where the observer asks the user a standard set of questions, using both the user's response and his / her own observation to answer the questions. If the purpose of the system is to relay (rather than extract) information to the human, then we could assign tests to assess whether the users remember what they were supposed to have learned. In an informal education scenario (e.g., a museum visit), we could plot these results against use satisfaction, so that the user experience is evaluated positively if it is both interesting/pleasant and educational. Another way could be to set up a task/game for the user, where in order to win it would be crucial to use information that only the ADS has. Then, we can evaluate the ADS based on it having successfully relayed the critical information to the human.

7.3 Conclusion

In this chapter, we have presented recent advances in the field of ADS, focusing mainly on adaptive DM. We also presented our thoughts on attributes we believe are missing from intelligent DS and some initial ideas on how we can achieve them. We view automatic feedback estimation, complex action learning, goal achievement, artificial social intelligence as well as user simulators as capabilities or affordances that can be built on top of an adaptive DM, that acts as the core technology. Moreover, we attempted to present ways that adaptive DM techniques can help push the state of the art in the aforementioned areas.

Artificial social intelligence, cooperative task completion, learning from demonstration are, in our opinion, essential attributes of the next generation ADS. As researchers have developed mature, robust and reliable methods for DM, NLG, ASR, NLU, etc., we can now afford to focus on higher level behaviour of ADS, such as automatically inferring what the task is, abstractly planning how to achieve it, or engaging in intricate discourse while exhibiting humour, sarcasm, laughter etc. at socially appropriate moments.

7.4 Impact On Society

We conclude this Ph.D. thesis with a note on the possible impact ADS and ALE, and specifically the work presented here, can have on our society. This can be best highlighted by possible applications, such as the ones presented in the Chapters 5 and 6, but in this section we attempt to provide some insight on how our contributions can be applied. One of the greatest benefits of adopting ADS in ALE is allowing technology not designed for ALE, to be accessible by ALE users, through an ADS. The proposed techniques are also a step towards robotic household assistants, as they make the system appear intelligent, by exhibiting complex behaviour, and, therefore, the user trusts it more and we are able to get much more information. This can lead to increased acceptance of ADS by the society, which in turn can also have a great impact on industry, as such systems are extremely cost effective.

REFERENCES

[1] A. Papangelis and F. Makedon, “Achieving complex behaviour in adaptive dialogue systems,” in *7th Metroplex Day*, 2013.

[2] A. Papangelis, N. Kouroupas, V. Karkaletsis, and F. Makedon, “An adaptive dialogue system with online dialogue policy learning,” in *Proceedings of the 7th Hellenic conference on Artificial Intelligence: theories and applications*, ser. SETN’12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 323–330.

[3] NIMH. National institute of mental health. [Online]. Available: http://www.nimh.nih.gov/index.shtml

[4] S. Janarthanam and O. Lemon, “A two-tier user simulation model for reinforcement learning of adaptive referring expression generation policies.” in *SIGDIAL Conference’09*, 2009, pp. 120–123.

[5] O. Lemon, “Learning what to say and how to say it: Joint optimisation of spoken dialogue management and natural language generation,” *Computer Speech & Language*, vol. 25, no. 2, pp. 210 – 221, 2011, language and speech issues in the engineering of companionable dialogue systems.

[6] C. Boidin, V. Rieser, L. Van Der Plas, L. Oliver, and J. Chevelu, *Predicting how it sounds: Re-ranking dialogue prompts based on TTS quality for adaptive Spoken Dialogue Systems*. ISCA, 2009, pp. 2487–2490. [Online]. Available: http://www.interspeech2009.org/conference/programme/session.php?id=6510

[7] M. Vacher, F. Portet, A. Fleury, and N. Noury, “Development of audio sensing technology for ambient assisted living: Applications and challenges,” *Interna-*

tional Journal of E-Health and Medical Communications, vol. 2, no. 1, pp. 35 – 54, march 2011.

[8] H. Sun, V. D. Florio, N. Gui, and C. Blondia, "Promises and challenges of ambient assisted living systems," in *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, ser. ITNG '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1201–1207.

[9] T. Greenhalgh, J. Wherton, P. Sugarhood, S. Hinder, R. Procter, and R. Stones, "What matters to older people with assisted living needs? a phenomenological analysis of the use and non-use of telehealth and telecare," *Social Science & Medicine*, vol. 93, no. 0, pp. 86 – 94, 2013.

[10] C. Harrefors, K. Axelsson, and S. Sävenstedt, "Using assistive technology services at differing levels of care: healthy older couples perceptions," *Journal of advanced nursing*, vol. 66, no. 7, pp. 1523–1532, 2010.

[11] M. Alwan, S. Dalal, D. Mack, S. Kell, B. Turner, J. Leachtenauer, and R. Felder, "Impact of monitoring technology in assisted living: outcome pilot," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 10, no. 1, pp. 192–198, 2006.

[12] M. Alwan, J. Leachtenauer, S. Dalal, D. Mack, S. Kell, B. Turner, and R. Felder, "Psychosocial impact of monitoring technology in assisted living: A pilot study," in *Information and Communication Technologies, 2006. ICTTA'06. 2nd*, vol. 1. IEEE, 2006, pp. 998–1002.

[13] V. Fuchsberger, "Ambient assisted living: elderly people's needs and how to face them," in *Proceedings of the 1st ACM international workshop on Semantic ambient media experiences.* ACM, 2008, pp. 21–24.

[14] E. M. Agree and V. A. Freedman, “A quality-of-life scale for assistive technology: results of a pilot study of aging and technology,” *Physical Therapy*, vol. 91, no. 12, pp. 1780–1788, 2011.

[15] H. Aloulou, M. Mokhtari, T. Tiberghien, J. Biswas, and P. Yap, “An adaptable and flexible framework for assistive living of cognitively impaired people,” 2013.

[16] H. Aloulou, M. Mokhtari, T. Tiberghien, J. Biswas, C. Phua, J. H. K. Lin, and P. Yap, “Deployment of assistive living technology in a nursing home environment: methods and lessons learned,” *BMC medical informatics and decision making*, vol. 13, no. 1, p. 42, 2013.

[17] G. Virone, M. Alwan, S. Dalal, S. W. Kell, B. Turner, J. A. Stankovic, and R. Felder, “Behavioral patterns of older adults in assisted living,” *Information Technology in Biomedicine, IEEE Transactions on*, vol. 12, no. 3, pp. 387–398, 2008.

[18] T. Kleinberger, M. Becker, E. Ras, A. Holzinger, and P. Müller, “Ambient intelligence in assisted living: enable elderly people to handle future interfaces,” in *Universal access in human-computer interaction. Ambient interaction.* Springer, 2007, pp. 103–112.

[19] J. Fasola and M. J. Matarić, “Using spatial language to guide and instruct robots in household environments,” in *2012 AAAI Fall Symposium Series*, 2012.

[20] D. A. Lazewatsky and W. D. Smart, “Context-sensitive in-the-world interfaces for mobile manipulation robots,” in *RO-MAN, 2012 IEEE.* IEEE, 2012, pp. 989–994.

[21] R. Fukazawa, K. Takashima, G. Shoemaker, Y. Kitamura, Y. Itoh, and F. Kishino, “Comparison of multimodal interactions in perspective-corrected multi-display environment,” in *3D User Interfaces (3DUI), 2010 IEEE Symposium on.* IEEE, 2010, pp. 103–110.

[22] P. Deegan, R. Grupen, A. Hanson, E. Horrell, S. Ou, E. Riseman, S. Sen, B. Thibodeau, A. Williams, and D. Xie, "Mobile manipulators for assisted living in residential settings," *Autonomous Robots*, vol. 24, no. 2, pp. 179–192, 2008.

[23] J. M. Beer and L. Takayama, "Mobile remote presence systems for older adults: acceptance, benefits, and concerns," in *Proceedings of the 6th international conference on Human-robot interaction*, ser. HRI '11. New York, NY, USA: ACM, 2011, pp. 19–26. [Online]. Available: http://doi.acm.org/10.1145/1957656.1957665

[24] A. Cesta, S. Coradeschi, G. Cortellessa, J. Gonzalez, L. Tiberio, and S. Von Rump, "Enabling social interaction through embodiment in excite," in *ForItAAL. Second Italian forum on ambient assisted living, Trento, October*, 2010, pp. 5–7.

[25] S. Coradeschi, A. Kristoffersson, A. Loutfi, S. Von Rump, A. Cesta, G. Cortellessa, and J. Gonzalez, "Towards a methodology for longitudinal evaluation of social robotic telepresence for elderly," in *1st Workshop on Social Robotic Telepresence at HRI 2011*, 2011.

[26] D. Labonte, F. Michaud, P. Boissy, H. Corriveau, R. Cloutier, and M.-A. Roux, "A pilot study on teleoperated mobile robots in home environments," in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 4466–4471.

[27] L. Tiberio, L. Padua, A. R. Pellegrino, I. Aprile, G. Cortellessa, and A. Cesta, "Assessing the tolerance of a telepresence robot in users with mild cognitive impairment," in *HRI 2011 Workshop*, vol. 2, 2011, p. 23.

[28] P. Boissy, H. Corriveau, F. Michaud, D. Labonté, and M.-P. Royer, "A qualitative study of in-home robotic telepresence for home care of community-living

elderly subjects," *Journal of Telemedicine and Telecare*, vol. 13, no. 2, pp. 79–84, 2007.

[29] V. Faucounau, Y.-H. Wu, M. Boulay, M. Maestrutti, and A.-S. Rigaud, "Caregivers' requirements for in-home robotic agent for supporting community-living elderly subjects with cognitive impairment," *Technology and Health Care*, vol. 17, no. 1, pp. 33–40, 2009.

[30] C. McMurrough, I. Ranatunga, A. Papangelis, D. O. Popa, and F. Makedon, "A development and evaluation platform for non-tactile power wheelchair controls," in *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments.* ACM, 2013, p. 4.

[31] G. Virone, A. Wood, L. Selavo, Q. Cao, L. Fang, T. Doan, Z. He, R. Stoleru, S. Lin, and J. Stankovic, "An assisted living oriented information system based on a residential wireless sensor network," in *Distributed Diagnosis and Home Healthcare, 2006. D2H2. 1st Transdisciplinary Conference on.* IEEE, 2006, pp. 95–100.

[32] A. Wood, G. Virone, T. Doan, Q. Cao, L. Selavo, Y. Wu, L. Fang, Z. He, S. Lin, and J. Stankovic, "Alarm-net: Wireless sensor networks for assisted-living and residential monitoring," *University of Virginia Computer Science Department Technical Report*, vol. 2, 2006.

[33] A. Wood, J. Stankovic, G. Virone, L. Selavo, Z. He, Q. Cao, T. Doan, Y. Wu, L. Fang, and R. Stoleru, "Context-aware wireless sensor networks for assisted living and residential monitoring," *Network, IEEE*, vol. 22, no. 4, pp. 26–33, 2008.

[34] J. M. Eklund, T. R. Hansen, J. Sprinkle, and S. Sastry, "Information technology for assisted living at home: building a wireless infrastructure for assisted living,"

in *Engineering in Medicine and Biology Society, 2005. IEEE-EMBS 2005. 27th Annual International Conference of the.* IEEE, 2006, pp. 3931–3934.

[35] J. Singh, J. Bacon, and K. Moody, “Dynamic trust domains for secure, private, technology-assisted living,” in *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on.* IEEE, 2007, pp. 27–34.

[36] S. Dagtas, Y. Natchetoi, and H. Wu, “An integrated wireless sensing and mobile processing architecture for assisted living and healthcare applications,” in *Proceedings of the 1st ACM SIGMOBILE international workshop on Systems and networking support for healthcare and assisted living environments.* ACM, 2007, pp. 70–72.

[37] K. Perakis, M. Haritou, and D. Koutsouris, “Aladdin, a technology platform for the assisted living of dementia elderly individuals and their carers,” in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living.* Springer, 2009, pp. 878–881.

[38] D. D. Vergados, “Service personalization for assistive living in a mobile ambient healthcare-networked environment,” *Personal and Ubiquitous Computing*, vol. 14, no. 6, pp. 575–590, 2010.

[39] A. Wigfield, S. Moore, C. Buse, and G. Fry, “Workforce development for assisted living technology: understanding roles, delivery and workforce needs,” 2012.

[40] J. Nehmer, M. Becker, A. Karshmer, and R. Lamm, “Living assistance systems: an ambient intelligence approach,” in *Proceedings of the 28th international conference on Software engineering.* ACM, 2006, pp. 43–50.

[41] C. N. Doukas and I. Maglogiannis, “Emergency fall incidents detection in assisted living environments utilizing motion, sound, and visual perceptual components,” *Information Technology in Biomedicine, IEEE Transactions on*, vol. 15, no. 2, pp. 277–289, 2011.

[42] Q. Wang, W. Shin, X. Liu, Z. Zeng, C. Oh, B. K. AlShebli, M. Caccamo, C. A. Gunter, E. Gunter, J. Hou, *et al.*, "I-living: An open system architecture for assisted living," in *Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference on*, vol. 5. IEEE, 2006, pp. 4268–4275.

[43] A. J. Jara, M. A. Zamora, and A. F. Skarmeta, "An internet of things—based personal device for diabetes therapy management in ambient assisted living (aal)," *Personal and Ubiquitous Computing*, vol. 15, no. 4, pp. 431–440, 2011.

[44] K. Kasugai, M. Ziefle, C. Röcker, and P. Russell, "Creating spatio-temporal contiguities between real and virtual rooms in an assistive living environment," *Proceedings of CREATE*, vol. 10, pp. 62–67, 2010.

[45] E. Levin, R. Pieraccini, and W. Eckert, "Learning dialogue strategies within the markov decision process framework," in *Automatic Speech Recognition and Understanding, 1997. Proceedings., 1997 IEEE Workshop on*, Dec. 1997, pp. 72 –79.

[46] A. Farinelli, A. Rogers, A. Petcu, and N. R. Jennings, "Decentralised coordination of low-power embedded devices using the max-sum algorithm," in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*. International Foundation for Autonomous Agents and Multiagent Systems, 2008, pp. 639–646.

[47] W. Swartout, D. Traum, R. Artstein, D. Noren, P. Debevec, K. Bronnenkant, J. Williams, A. Leuski, S. Narayanan, D. Piepol, C. Lane, J. Morie, P. Aggarwal, M. Liewer, J.-Y. Chiang, J. Gerten, S. Chu, and K. White, "Ada and grace: toward realistic and engaging virtual museum guides," in *Proceedings of the 10th international conference on Intelligent virtual agents*, ser. IVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 286–300. [Online]. Available: http://dl.acm.org/citation.cfm?id=1889075.1889110

[48] R. D. F. Gnjatovic, Milan, "On the role of the nimitek corpus in developing an emotion adaptive spoken dialogue system," ser. LREC, 2008.

[49] H. Cuayáhuitl, S. Renals, O. Lemon, and H. Shimodaira, "Evaluation of a hierarchical reinforcement learning spoken dialogue system," *Comput. Speech Lang.*, vol. 24, pp. 395–429, April 2010.

[50] D. Traum and S. Larsson, "The information state approach to dialogue management," in *Current and New Directions in Discourse and Dialogue*, J. van Kuppevelt and R. Smith, Eds. Dordrecht, the Netherlands: Kluwer Academic Publishers, 2003.

[51] S. Young, M. Gasic, S. Keizer, F. Mairesse, J. Schatzmann, B. Thomson, and K. Yu, "The hidden information state model: A practical framework for pomdp-based spoken dialogue management," *Computer Speech & Language*, vol. 24, no. 2, pp. 150 – 174, 2010.

[52] J. Henderson and O. Lemon, "Mixture model pomdps for efficient handling of uncertainty in dialogue management," in *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, ser. HLT-Short '08. Stroudsburg, PA, USA: Association for Computational Linguistics, 2008, pp. 73–76. [Online]. Available: http://portal.acm.org/citation.cfm?id=1557690.1557710

[53] T. H. Bui, J. Zwiers, M. Poel, and A. Nijholt, "Affective dialogue management using factored pomdps," in *Interactive Collaborative Information Systems*, 2010, pp. 207–236.

[54] T. Bui, M. Poel, A. Nijholt, and J. Zwiers, "A tractable hybrid ddn-pomdp approach to affective dialogue modeling for probabilistic frame-based dialogue systems," in *BNAIC 2008, Proceedings 20th Belgian-Netherlands Conference on Artificial Intelligence*, A. Nijholt, M. Pantic, M. Poel, and G. Hondorp, Eds.

Enschede: University of Twente, October 2008, pp. 289–290, this is an extended abstract of a paper that has been acepted for publication in Journal of Natural Language Engineering.

[55] K. Kanazawa and T. L. Dean, "A model for projection and action." in *IJCAI*, vol. 89, 1989, pp. 985–990.

[56] N. Webb, D. Benyon, J. Bradley, P. Hansen, and O. Mival, "Wizard of oz experiments for a companion dialogue system: Eliciting companionable conversation," in *Proceedings of the Seventh conference on International Language Resources and Evaluation (LREC'10)*, N. C. C. Chair), K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, and D. Tapias, Eds. Valletta, Malta: European Language Resources Association (ELRA), may 2010.

[57] B. Reeves and C. Nass, *The Media equation: how people treat computers, television, and new media*. Cambridge University Press, 1997.

[58] G. Sieber and B. Krenn, "Towards an Episodic Memory for Companion Dialogue," in *Intelligent Virtual Agents*, ser. Lecture Notes in Computer Science, J. Allbeck, N. Badler, T. Bickmore, C. Pelachaud, and A. Safonova, Eds. Berlin, Heidelberg: Springer Berlin / Heidelberg, 2010, vol. 6356, ch. 33, pp. 322–328–328. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-15892-6_33

[59] Y. Wilks, R. Catizone, S. Worgan, A. Dingli, R. Moore, D. Field, and W. Cheng, "A prototype for a conversational companion for reminiscing about images." *Computer Speech & Language*.

[60] C. Adam, L. Cavedon, and L. Padgham, ""hello emily, how are you today?": personalised dialogue in a toy to engage children," in *Proceedings of the 2010 Workshop on Companionable Dialogue Systems*, ser. CDS '10. Stroudsburg,

PA, USA: Association for Computational Linguistics, 2010, pp. 19–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870559.1870563

[61] ——, "Flexible conversation management in an engaging virtual character," *International Workshop on Interacting with ECAs as Virtual Characters*, p. 1, 2010.

[62] H. Cuayáhuitl and N. Dethlefs, "Hierarchical multiagent reinforcement learning for coordinating verbal and non-verbal actions in robots," *Machine Learning for Interactive Systems: Bridging the Gap Between Language, Motor*, p. 27, 2012.

[63] A. Robotics. (2013, Nov.) Nao robot. [Online]. Available: http://www.aldebaran-robotics.com/en/

[64] I. Kruijff-Korbayová, H. Cuayáhuitl, B. Kiefer, M. Schröder, P. Csi, G. Paci, G. Sommavilla, F. Tesser, H. Sahli, G. Athanasopoulos, *et al.*, "A conversational system for multi-session child-robot interaction with several games," in *German Conference on Artificial Intelligence (KI), Saarbruecken, Germany*, 2012.

[65] G.-J. Kruijff and H.-U. Krieger, "Learnable controllers for adaptive dialogue processing management," in *Proceedings of the AAAI 2010 Fall Symposium "Dialogue with Robots"*, November 2010.

[66] R. Meena, G. Skantze, and J. Gustafson, "A data-driven approach to understanding spoken route directions in human-robot dialogue." in *INTERSPEECH*, 2012.

[67] S. Tellex, P. Thaker, R. Deits, T. Kollar, and N. Roy, "Toward information theoretic human-robot dialog." in *Robotics: Science and Systems*, 2012.

[68] M. Q. Azhar, S. Parsons, and E. Sklar, "An argumentation-based dialogue system for human-robot collaboration," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems.* Interna-

tional Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1353–1354.

[69] M. Giuliani, M. E. Foster, A. Isard, C. Matheson, J. Oberlander, and A. Knoll, "Situated reference in a hybrid human-robot interaction system," in *Proceedings of the 6th International Natural Language Generation Conference*, ser. INLG '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 67–75. [Online]. Available: http://portal.acm.org/citation.cfm?id=1873738.1873749

[70] M. E. Foster, M. Giuliani, A. Isard, C. Matheson, J. Oberlander, and A. Knoll, "Evaluating description and reference strategies in a cooperative human-robot dialogue system." in *IJCAI*, 2009, pp. 1818–1823.

[71] G.-J. Kruijff, "Trusting in human-robot teams given asymmetric agency and social sentience," in *Proceedings of the 2013 AAAI Spring Symposium on Trust and Autonomous Systems*, AAAI. Stanford University: AAAI Press, 3 2013.

[72] M. Ochs and C. Pelachaud, "Model of the perception of smiling virtual character," in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*, ser. AAMAS '12. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 87–94.

[73] E. Bevacqua, S. Pammi, S. J. Hyniewska, M. Schröder, and C. Pelachaud, "Multimodal backchannels for embodied conversational agents," in *Proceedings of the 10th international conference on Intelligent virtual agents*, ser. IVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 194–200.

[74] E. De Sevin, S. J. Hyniewska, and C. Pelachaud, "Influence of personality traits on backchannel selection," in *Proceedings of the 10th international conference*

on Intelligent virtual agents, ser. IVA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 187–193.

[75] K. Prepin and C. Pelachaud, "Effect of time delays on agents' interaction dynamics," in *The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 3*, ser. AAMAS '11. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 1055–1062.

[76] S. Konstantopoulos and V. Karkaletsis, "System personality and adaptivity in affective human-computer interaction," *International Journal on Artificial Intelligence Tools*, vol. 22, no. 02, p. 1350014, 2013.

[77] K. Bergmann, H. Rieser, and S. Kopp, "Regulating dialogue with gestures—towards an empirically grounded simulation with conversational agents," ser. Proceedings of the SIGdial 2011 Conference, J. Chai, J. Moore, R. Passonneau, and D. Traum, Eds. Association for Computational Linguistics, 2011, pp. 88–97.

[78] K. Bergmann and S. Kopp, "Gestural alignment in natural dialogue," ser. Proceedings of the 34th Annual Conference of the Cognitive Science Society (CogSci 2012), R. P. Cooper, D. Peebles, and N. Miyake, Eds. Cognitive Sciece Society, 2012, pp. 1326–1331.

[79] A. Vinciarelli, M. Pantic, D. Heylen, C. Pelachaud, I. Poggi, F. D'Errico, and M. Schroeder, "Bridging the gap between social animal and unsocial machine: A survey of social signal processing," *Affective Computing, IEEE Transactions on*, vol. 3, no. 1, pp. 69–87, 2012.

[80] D. Bohus and A. I. Rudnicky, "The ravenclaw dialog management framework: Architecture and systems," *Computer Speech & Language*, vol. 23, no. 3, pp. 332 – 361, 2009.

[81] M. Gasic, F. Jurcicek, S. Keizer, F. Mairesse, B. Thomson, K. Yu, and S. Young, “Gaussian processes for fast policy optimisation of pomdp-based dialogue managers,” in *Proceedings of the SIGDIAL 2010 Conference*, Association for Computational Linguistics. Tokyo, Japan: Association for Computational Linguistics, September 2010, p. 201–204. [Online]. Available: http://www.aclweb.org/anthology/W/W10/W10-4334

[82] O. Pietquin, M. Geist, S. Chandramohan, and H. Frezza-Buet, “Sample-Efficient Batch Reinforcement Learning for Dialogue Management Optimization,” *ACM Transactions on Speech and Language Processing*, vol. 7, no. 3, pp. 7:1–7:21, May 2011.

[83] F. Jurčíček, B. Thomson, S. Keizer, F. Mairesse, M. Gašić, K. Yu, and S. Young, “Natural belief-critic: A reinforcement algorithm for parameter estimation in statistical spoken dialogue systems,” in *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[84] D. S. Oliver Lemon, Xingkun Liu and C. Tollander, “Hierarchical reinforcement learning of dialogue policies in a development environment for dialogue systems: Reall-dude,” ser. SemDial Workshop on the Semantics and Pragmatics of Dialogue, 2006.

[85] V. Rieser and O. Lemon, “Does this list contain what you were searching for? learning adaptive dialogue strategies for interactive question answering,” *Nat. Lang. Eng.*, vol. 15, pp. 55–72, January 2009.

[86] P. A. Crook and O. Lemon, “Accurate probability estimation of hypothesised user acts for POMDP approaches to dialogue management,” in *Proceedings of the Twelfth Annual Research Colloquium of the Special-Interest Group for Computational Linguistics in the UK and Ireland (CLUKI)*, 2009.

[87] ——, “Representing uncertainty about complex user goals in statistical dialogue systems,” in *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, ser. SIGDIAL ’10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 209–212. [Online]. Available: http://portal.acm.org/citation.cfm?id=1944506.1944541

[88] ——, “Lossless Value Directed Compression of Complex User Goal States for Statistical Spoken Dialogue Systems,” in *Proceedings of the Twelfth Annual Conference of the International Speech Communication Association (Interspeech)*, August 2011.

[89] F. Morbini, D. DeVault, K. Sagae, J. Gerten, A. Nazarian, and D. Traum, “FLoReS: a forward looking, reward seeking, dialogue manager,” in *4th International Workshop on Spoken Dialog Systems*, Paris, France, Nov. 2012.

[90] B. Thomson and S. Young, “Bayesian update of dialogue state: A pomdp framework for spoken dialogue systems,” *Computer Speech & Language*, vol. 24, no. 4, pp. 562 – 588, 2010.

[91] C. Bishop *et al.*, *Pattern recognition and machine learning.* springer New York, 2006.

[92] J. Peters, S. Vijayakumar, and S. Schaal, “Natural actor-critic,” *Machine Learning: ECML 2005*, pp. 280–291, 2005.

[93] H. Cuayáhuitl and N. Dethlefs, “Dialogue systems using online learning: Beyond empirical methods,” in *NAACL-HLT Workshop on Future directions and needs in the Spoken Dialog Community: Tools and Data (SDCTD 2012).* Montréal, Canada: Association for Computational Linguistics, June 2012, pp. 7–8.

[94] N. Dethlefs, H. Hastie, V. Rieser, and O. Lemon, “Optimising incremental dialogue decisions using information density for interactive systems,” in *Proceed-*

ings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning. Association for Computational Linguistics, 2012, pp. 82–93.

[95] D. Traum, D. DeVault, J. Lee, Z. Wang, and S. Marsella, "Incremental dialogue understanding and feedback for multiparty, multimodal conversation," in *Proceedings of the 12th international conference on Intelligent Virtual Agents*, ser. IVA'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 275–288.

[96] L. E. Asri, R. Laroche, and O. Pietquin, "Reward Function Learning for Dialogue Management," in *Proceedings of the sixth Starting Artificial Intelligence Research Symposium (STAIRS 2012)*, Montpellier (France), August 2012, pp. 95 – 106.

[97] M. Frampton and O. Lemon, "Recent research advances in reinforcement learning in spoken dialogue systems," *The Knowledge Engineering Review*, vol. 24, no. 04, pp. 375–408, 2009.

[98] H. Cuayáhuitl and N. Dethlefs, "Spatially-aware dialogue control using hierarchical reinforcement learning," *ACM Trans. Speech Lang. Process.*, vol. 7, pp. 5:1–5:26, June 2011. [Online]. Available: http://doi.acm.org/10.1145/1966407.1966410

[99] M. A. Walker, D. J. Litman, C. A. Kamm, A. A. Kamm, and A. Abella, "Paradise: A framework for evaluating spoken dialogue agents," 1997, pp. 271–280.

[100] A. Hof, E. Hagen, and A. Huber, "Adaptive help for speech dialogue systems based on learning and forgetting of speech commands," in *Proceedings of the 7th SIGdial Workshop on Discourse and Dialogue*, ser. SigDIAL '06. Stroudsburg, PA, USA: Association for Computational Linguistics, 2006, pp. 1–8. [Online]. Available: http://portal.acm.org/citation.cfm?id=1654595.1654597

[101] J. Henderson, O. Lemon, and K. Georgila, "Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets," *Comput. Linguist.*, vol. 34, pp. 487–511, December 2008. [Online]. Available: http://dx.doi.org/10.1162/coli.2008.07-028-R2-05-82

[102] P. Lison, "Towards relational pomdps for adaptive dialogue management," in *Proceedings of the ACL 2010 Student Research Workshop*, ser. ACLstudent '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 7–12. [Online]. Available: http://portal.acm.org/citation.cfm?id=1858913.1858915

[103] M. Gasic, F. Jurcícek, B. Thomson, K. Yu, and S. Young, "On-line policy optimisation of spoken dialogue systems via live interaction with human subjects," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on.* IEEE, 2011, pp. 312–317.

[104] M. Gašic, C. Breslin, M. Henderson, D. Kim, M. Szummer, B. Thomson, P. Tsiakoulis, and S. Young, "On-line policy optimisation of bayesian spoken dialogue systems via human interaction."

[105] ——, "Pomdp-based dialogue manager adaptation to extended domains."

[106] V. Rieser and O. Lemon, "Using machine learning to explore human multimodal clarification strategies," in *Proceedings of the COLING/ACL on Main conference poster sessions*, ser. COLING-ACL '06. Stroudsburg, PA, USA: Association for Computational Linguistics, 2006, pp. 659–666. [Online]. Available: http://dl.acm.org/citation.cfm?id=1273073.1273158

[107] M. Geist and O. Pietquin, "Kalman temporal differences," *J. Artif. Int. Res.*, vol. 39, pp. 483–532, September 2010. [Online]. Available: http://dl.acm.org/citation.cfm?id=1946417.1946428

[108] N. Kimura, C. Hori, T. Misu, K. Ohtake, H. Kawai, and S. Nakamura, “Expansion of wfst-based dialog management for handling multiple asr hypotheses,” in *Proceedings of the Second international conference on Spoken dialogue systems for ambient environments*, ser. IWSDS’10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 61–72. [Online]. Available: http://portal.acm.org/citation.cfm?id=1925948.1925955

[109] H. Cuayhuitl, S. Renals, O. Lemon, and H. Shimodaira, “Hierarchical dialogue optimization using semi-markov decision processes,” in *In Proceedings of the European Conference on Speech Communication and Technologies (Interspeech07), Anvers*, 2007.

[110] L. Q. Anh and C. Pelachaud, “Expressive gesture model for humanoid robot,” in *Proceedings of the 4th international conference on Affective computing and intelligent interaction - Volume Part II*, ser. ACII’11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 224–231.

[111] ——, “Generating co-speech gestures for the humanoid robot nao through bml,” in *Proceedings of the 9th international conference on Gesture and Sign Language in Human-Computer Interaction and Embodied Communication*, ser. GW’11. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 228–237.

[112] S. Chandramohan, M. Geist, and O. Pietquin, “Optimizing Spoken Dialogue Management with Fitted Value Iteration,” in *Proceedings of the International Conference on Speech Communication and Technologies (Interspeech 2010)*. Makuhari (Japan): ISCA, September 2010, pp. 86–89.

[113] M. G. Lagoudakis and R. Parr, “Least-squares policy iteration,” *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[114] S. Chandramohan and O. Pietquin, “User and noise adaptive dialogue management using hybrid system actions,” in *Proceedings of the Second*

international conference on Spoken dialogue systems for ambient environments, ser. IWSDS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 13–24. [Online]. Available: http://portal.acm.org/citation.cfm?id=1925948.1925951

[115] F. Jurvčíček, B. Thomson, and S. Young, "Natural actor and belief critic: Reinforcement algorithm for learning parameters of dialogue systems modelled as pomdps," *ACM Transactions on Speech and Language Processing (TSLP)*, vol. 7, no. 3, p. 6, 2011.

[116] S. Moller, R. Englert, K. Engelbrecht, V. Hafner, A. Jameson, A. Oulasvirta, E. Raake, and N. Reithinger, "Memo: Towards automatic usability evaluation of spoken dialogue services by user error simulations," in *InterSpeech*, 2006, pp. 1786–1789.

[117] K.-P. Engelbrecht, F. Godde, F. Hartard, H. Ketabdar, and S. Moller, "Modeling user satisfaction with hidden markov model," in *Proceedings of the SIGDIAL 2009 Conference: The 10th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, ser. SIGDIAL '09. Stroudsburg, PA, USA: Association for Computational Linguistics, 2009, pp. 170–177. [Online]. Available: http://portal.acm.org/citation.cfm?id=1708376.1708402

[118] S. Rossignol, O. Pietquin, and M. Ianotto, "Simulation of the grounding process in spoken dialog systems with bayesian networks," in *Proceedings of the Second international conference on Spoken dialogue systems for ambient environments*, ser. IWSDS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 110–121. [Online]. Available: http://portal.acm.org/citation.cfm?id=1925948.1925959

[119] S. Chandramohan, M. Geist, F. Lefèvre, and O. Pietquin, "User Simulation in Dialogue Systems using Inverse Reinforcement Learning," in *Proceedings of the 12th Annual Conference of the International Speech Communication Association (Interspeech 2011)*, Florence (Italy), August 2011, pp. 1025–1028.

[120] S. Chandramohan, M. Geist, F. Lefèvre, and O. Pietquin, “Behavior specific user simulation in spoken dialogue systems,” in *Speech Communication; 10. ITG Symposium; Proceedings of.* VDE, 2012, pp. 1–4.

[121] S. Chandramohan, M. Geist, F. Lefevre, O. Pietquin, M.-I. Supelec, and F. Metz, “Co-adaptation in spoken dialogue systems,” *Proceedings of the Fourth International Workshop on Spoken Dialog Systems*, 2012.

[122] S. Keizer, M. Gašić, F. Jurčíček, F. Mairesse, B. Thomson, K. Yu, and S. Young, “Parameter estimation for agenda-based user simulation,” in *Proceedings of the 11th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, ser. SIGDIAL ’10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 116–123. [Online]. Available: http://dl.acm.org/citation.cfm?id=1944506.1944529

[123] J. Schatzmann and S. Young, “The hidden agenda user simulation model,” *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 17, no. 4, pp. 733 –747, may 2009.

[124] J. Schatzmann, B. Thomson, and S. Young, “Statistical User Simulation with a Hidden Agenda,” in *8th SIGDial Workshop on Discourse and Dialogue*, Antwerp, Belgium, Sept. 2007.

[125] J. Schatzmann, K. Georgila, and S. Young, “Quantitative evaluation of user simulation techniques for spoken dialogue systems,” in *IN PROC. OF 6TH SIGDIAL*, 2005, pp. 45–54.

[126] M. Hartikainen, E. pekka Salonen, and M. Turunen, “Subjective evaluation of spoken dialogue systems using servqual method,” in *Proceedings of the Eighth International Conference on Spoken Language Processing (INTERSPEECH 2004-ICSLP), Jeju Island, Korea*, 2004.

[127] S. Moller, P. Smeele, H. Boland, and J. Krebber, "Evaluating spoken dialogue systems according to de-facto standards: A case study," *Computer Speech & Language*, vol. 21, no. 1, pp. 26 – 53, 2007.

[128] E. Mizukami, H. Kashioka, H. Kawai, and S. Nakamura, "A study toward an evaluation method for spoken dialogue systems considering user criteria," in *Proceedings of the Second international conference on Spoken dialogue systems for ambient environments*, ser. IWSDS'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 176–181. [Online]. Available: http://portal.acm.org/citation.cfm?id=1925948.1925968

[129] O. Pietquin and H. Hastie, "A survey on metrics for the evaluation of user simulations," *The Knowledge Engineering Review*, vol. 28, pp. 59–73, 2 2013.

[130] D. DeVault, A. Leuski, and K. Sagae, "Toward learning and evaluation of dialogue policies with text examples," in *Proceedings of the SIGDIAL 2011 Conference*. Portland, Oregon: Association for Computational Linguistics, June 2011, pp. 39–48. [Online]. Available: http://www.aclweb.org/anthology/W/W11/W11-2006

[131] A. Papangelis, V. Metsis, J. Shawe-Taylor, and F. Makedon, "Sensor placement and coordination via distributed multi-agent cooperative control," in *Proceedings of the 3rd PETRAE*. ACM, 2010, pp. 14:1–14:8.

[132] A. Papangelis and F. Makedon, "A tool for sensor placement and system monitoring in assistive environments," in *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '11. New York, NY, USA: ACM, 2011, pp. 31:1–31:4.

[133] G. Galatas, A. Papangelis, and F. Makedon, "A framework for optimal assistive robot placement for event recognition," in *ICNC*, 2013.

[134] A. Papangelis, G. Galatas, and F. Makedon, "A recommender system for assistive environments," in *Proceedings of the 4th International Conference on PErvasive Technologies Related to Assistive Environments*, ser. PETRA '11. New York, NY, USA: ACM, 2011, pp. 6:1–6:4. [Online]. Available: http://doi.acm.org/10.1145/2141622.2141630

[135] C. Szepesvári, "Algorithms for reinforcement learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 4, no. 1, pp. 1–103, 2010.

[136] J. Pineau, G. Gordon, S. Thrun, *et al.*, "Point-based value iteration: An anytime algorithm for pomdps," in *IJCAI*, vol. 3, 2003, pp. 1025–1032.

[137] M. T. Spaan and N. A. Vlassis, "Perseus: Randomized point-based value iteration for pomdps." *J. Artif. Intell. Res.(JAIR)*, vol. 24, pp. 195–220, 2005.

[138] T. Smith and R. Simmons, "Heuristic search value iteration for pomdps," in *Proceedings of the 20th conference on Uncertainty in artificial intelligence.* AUAI Press, 2004, pp. 520–527.

[139] M. Hauskrecht, "Incremental methods for computing bounds in partially observable markov decision processes," in *AAAI/IAAI.* Citeseer, 1997, pp. 734–739.

[140] S. Ross, J. Pineau, S. Paquet, and B. Chaib-draa, "Online planning algorithms for pomdps," *J. Artif. Int. Res.*, vol. 32, pp. 663–704, July 2008. [Online]. Available: http://dl.acm.org/citation.cfm?id=1622673.1622690

[141] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[142] A. G. Barto, "Recent advances in hierarchical reinforcement learning," vol. 13, p. 2003, 2003.

[143] S. Bhatnagar, R. S. Sutton, M. Ghavamzadeh, and M. Lee, "Incremental natural actor-critic algorithms," 2007.

[144] R. S. Sutton and A. G. Barto, “Reinforcement learning: An introduction,” *The MIT Press, Cambridge, MA*, 1998.

[145] A. Papangelis, V. Karkaletsis, and F. Makedon, “Evaluation of online dialogue policy learning techniques,” in *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, N. C. C. Chair), K. Choukri, T. Declerck, M. U. Doan, B. Maegaard, J. Mariani, J. Odijk, and S. Piperidis, Eds. Istanbul, Turkey: European Language Resources Association (ELRA), may 2012.

[146] A. Papangelis, “A comparative study of reinforcement learning techniques on dialogue management,” in *Proceedings of the Student Research Workshop at the 13th Conference of the European Chapter of the Association for Computational Linguistics*, ser. EACL '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 22–31. [Online]. Available: http://dl.acm.org/citation.cfm?id=2380943.2380946

[147] A. Papangelis, V. Karkaletsis, and F. Makedon, “Online complex action learning and user state estimation for adaptive dialogue systems,” in *Proc. ICTAI-12 24th IEEE International Conference On Tools With Artificial Intelligence*, 2012.

[148] A. Papangelis, I. Ranatunga, C. McMurrough, and F. Makedon, “An assistive object manipulation system,” in *Proceedings of the 3rd Workshop on Robotics for Assistive Environments (RasEnv)*, 2013, p. to appear.

[149] J. A. Russell, “A circumplex model of affect.” *Journal of Personality and Social Psychology*, vol. 39, no. 6, pp. 1161–1178, Dec. 1980.

[150] V. Rieser, *Reinforcement learning for adaptive dialogue systems.* Springerverlag Berlin Heidelberg, 2011.

[151] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *Proc. ICRA-09 Workshop on Open Source Software (ICRAOSS-09)*, 2009.

[152] D. Huggins-Daines, M. Kumar, A. Chan, A. Black, M. Ravishankar, and A. Rudnicky, "Pocketsphinx: A free, real-time continuous speech recognition system for hand-held devices," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, 2006, pp. I–I.

[153] M. Pickett and A. G. Barto, "Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning," in *ICML*, 2002, pp. 506–513.

[154] S. Konstantopoulos, "An embodied dialogue system with personality and emotions," in *Proceedings of the 2010 Workshop on Companionable Dialogue Systems*, ser. CDS '10. Stroudsburg, PA, USA: Association for Computational Linguistics, 2010, pp. 31–36. [Online]. Available: http://dl.acm.org/citation.cfm?id=1870559.1870565

[155] J. Peng and R. J. Williams, "Incremental multi-step q-learning," *Machine Learning*, vol. 22, no. 1-3, pp. 283–290, 1996.

[156] M. A. Wiering and H. Van Hasselt, "The qv family compared to other reinforcement learning algorithms," *2009 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning*, p. 8, 2009.

[157] O. Pietquin and H. Hastie, "A survey on metrics for the evaluation of user simulations," *The Knowledge Engineering Review*, vol. 28, pp. 59–73, 2 2013.

[158] F. Jurvčíček, B. Thomson, and S. Young, "Natural actor and belief critic: Reinforcement algorithm for learning parameters of dialogue systems modelled as pomdps," vol. 7, no. 3. ACM, 2011, p. 6.

[159] D. Traum and S. Larsson, “The information state approach to dialogue management,” in *Current and New Directions in Discourse and Dialogue*, J. van Kuppevelt and R. Smith, Eds. Dordrecht, the Netherlands: Kluwer Academic Publishers, 2003.

[160] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “ROS: an open-source Robot Operating System,” in *Proc. ICRA-09 Workshop on Open Source Software (ICRAOSS-09)*, 2009.

[161] R. Givan, T. Dean, and M. Greig, “Equivalence notions and model minimization in markov decision processes,” *Artif. Intell.*, vol. 147, no. 1-2, pp. 163–223, July 2003.

[162] A. Rizzo, K. Sagae, E. Forbell, J. Kim, B. Lange, J. Buckwalter, J. Williams, T. Parsons, P. Kenny, D. R. Traum, J. Difede, and B. Rothbaum, “Simcoach: An intelligent virtual human system for providing healthcare information and support,” in *I/ITSEC 2011*, Orlando, Florida, November 2011 2011.

[163] F. W. Weathers, B. T. Litz, D. S. Herman, J. A. Huska, T. M. Keane, *et al.*, “The ptsd checklist (pcl): Reliability, validity, and diagnostic utility,” in *annual meeting of the international society for traumatic stress studies, San Antonio, TX*, vol. 141, no. 7, 1993.

[164] S. Institute. [Online]. Available: http://www.sidran.org

[165] A. Rizzo, B. John, B. Newman, J. Williams, A. Hartholt, C. Lethin, and J. G. Buckwalter, “Virtual reality as a tool for delivering ptsd exposure therapy and stress resilience training,” *Military Behavioral Health*, vol. 1, no. 1, pp. 52–58, 2013.

[166] I. of Medicine, “Treatment for posttraumatic stress disorder in military and veteran populations: Initial assessment,” 2012.

[167] E. Foa, E. Hembree, and B. O. Rothbaum, *Prolonged exposure therapy for PTSD: Emotional processing of traumatic experiences therapist guide.* Oxford University Press, 2007.

[168] P. A. Resick, L. F. Williams, M. K. Suvak, C. M. Monson, and J. L. Gradus, "Long-term outcomes of cognitive–behavioral treatments for posttraumatic stress disorder among female rape survivors." *Journal of consulting and clinical psychology*, vol. 80, no. 2, p. 201, 2012.

[169] K. Kuch, B. J. Cox, and D. M. Direnfeld, "A brief self-rating scale for ptsd after road vehicle accident," *Journal of Anxiety Disorders*, vol. 9, no. 6, pp. 503–514, 1995.

[170] A. Papangelis, R. Gatchel, V. Metsis, and F. Makedon, "An adaptive dialogue system for assessing post traumatic stress disorder," in *Proceedings of the 6th International Conference on PErvasive Technologies Related to Assistive Environments.* ACM, 2013, p. 49.

[171] J. G. Buckwalter, J. Williams, K. Sagae, and D. Traum, "Simcoach an online intelligent virtual human agent system for breaking down barriers to care for service members and veterans," *Healing War Trauma: A Handbook of Creative Approaches*, vol. 44, p. 238, 2013.

[172] A. M. COOPER and R. Michels, "Diagnostic and statistical manual of mental disorders," *American Journal of Psychiatry*, vol. 138, no. 1, pp. 128–129, 1981.

[173] R. J. Kohlenberg and M. Tsai, "Healing interpersonal trauma with the intimacy of the therapeutic relationship." 1998.

[174] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems.* University of Cambridge, Department of Engineering, 1994.

[175] H. Cuayhuitl, “Hierarchical reinforcement learning for spoken dialogue systems,” Ph.D. dissertation, The University of Edinburgh, 2009. [Online]. Available: http://hdl.handle.net/1842/3852

[176] W. C. Chuang, Z.J, “Multi-modal emotion recognition from speech and text,” *Computational Linguistics and Chinese Language Processing*, vol. 9, no. 2, pp. 45–623, 2004.

[177] T. Drummond, “Vocabulary of emotions.”

[178] A. Gatt and E. Reiter, “Simplenlg: A realisation engine for practical applications,” in *Proceedings of the 12th European Workshop on Natural Language Generation*. Association for Computational Linguistics, 2009, pp. 90–93.

[179] R. Molich and J. Nielsen, “Improving a human-computer dialogue,” *Communications of the ACM*, vol. 33, no. 3, pp. 338–348, 1990.

[180] J. Nielsen and R. Molich, “Heuristic evaluation of user interfaces,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1990, pp. 249–256.

[181] J. Nielsen, “Enhancing the explanatory power of usability heuristics,” in *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*. ACM, 1994, pp. 152–158.

[182] ——, “Heuristic evaluation,” *Usability inspection methods*, vol. 17, pp. 25–62, 1994.

[183] H. Wang, H. Huang, and F. Makedon, “Emotion detection via discriminant laplacian embedding,” *Universal Access in the Information Society*, pp. 1–9.

[184] T. Paek, “Toward evaluation that leads to best practices: reconciling dialog evaluation in research and industry,” in *Proceedings of the Workshop on Bridging the Gap: Academic and Industrial Research in Dialog Technologies*, ser.

NAACL-HLT-Dialog ’07. Stroudsburg, PA, USA: Association for Computational Linguistics, 2007, pp. 40–47.

BIOGRAPHICAL STATEMENT

Alexandros Papangelis was born in Athens, Greece in 1986. He received his B.Sc. degree from the National and Kapodistrian University of Athens, in 2003, in Informatics and Telecommunications. He received, his M.Sc. from University College London, in 2009, in Machine Learning. His current research interest is in the area of adaptive dialogue management, robot dialogue systems and assistive living applications.