



An agent-based approach to dialogue management in personal assistants

Author:

Nguyen, Thi Thuc Anh

Publication Date:

2007

DOI:

<https://doi.org/10.26190/unsworks/17246>

License:

<https://creativecommons.org/licenses/by-nc-nd/3.0/au/>

Link to license to see what you are allowed to do with this resource.

Downloaded from <http://hdl.handle.net/1959.4/28241> in <https://unsworks.unsw.edu.au> on 2024-05-14

THE UNIVERSITY OF NEW SOUTH WALES

School of Computer Science and Engineering



An Agent-Based Approach to Dialogue Management in Personal Assistants

A THESIS SUBMITTED IN FULFILMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Nguyen, Thi Thuc Anh

December 2006

Supervisor: Wayne Wobcke

ORIGINALITY STATEMENT

‘I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project’s design and conception or in style, presentation and linguistic expression is acknowledged’

Abstract

Personal assistants need to allow the user to interact with the system in a flexible and adaptive way such as through spoken language dialogue. This research is aimed at achieving robust and effective dialogue management in such applications. We focus on an application, the *Spoken Personal Assistant* (SPA), in which the user can use a variety of devices to interact with a collection of personal assistants, each specializing in a task domain. The current implementation of the SPA contains an e-mail management agent and a calendar agent that the user can interact with through a spoken dialogue and a graphical interface on PDAs. The user-system interaction is handled by a Dialogue Manager agent.

We propose an agent-based approach that makes use of a BDI agent architecture for dialogue modelling and control. The Dialogue Manager agent of the SPA acts as the central point for maintaining coherent user-system interaction and coordinating the activities of the assistants. The dialogue model consists of a set of complex but modular plans for handling communicative goals. The dialogue control flow emerges automatically as the result of the agent's plan selection by the BDI interpreter. In addition the Dialogue Manager maintains the conversational context, the domain-specific knowledge and the user model in its internal beliefs.

We also consider the problem of dialogue adaptation in such agent-based dialogue systems. We present a novel way of integrating learning into a BDI architecture so that the agent can learn to select the most suitable plan among those applicable in the current context. This enables the Dialogue Manager agent to tailor its responses according to the conversational context and the user's physical context, devices and preferences.

Finally, we report the evaluation results, which indicate the robustness and effectiveness of the dialogue model in handling a range of users.

Preface

The research reported in this thesis is a part of the Smart Personal Assistant project, which was carried out under the Australian Cooperative Research Centre for Smart Internet Technology. The Smart Internet Technology CRC is an organization supported by the Australian government where universities and industry partners collaborate on applied research projects with a commercial orientation. The main research outcome of this project is the SPA spoken dialogue application, which allows users to remotely access and manage their e-mail and calendar through natural language dialogue and a graphical user interface on PDAs. The research contribution of the author in the Smart Personal Assistant project includes: design and development of the Dialogue Manager agent, the E-mail agent, the Calendar agent and the Partial Parser of the SPA application; and partial contribution to the design, development and evaluation of the SPA system as a whole.

Acknowledgments

Firstly, I am deeply grateful to my supervisor, Wayne Wobcke, for his valuable guidance and support throughout this research. Whenever I needed advice, I have always received his responses promptly. My work would have been much more difficult without help and encouragement from him.

I would like to express my earnest gratitude to my parents. I dedicate this thesis to them for their love, immense support and sacrifices over the many years that I have been away from home. Special thanks to my husband for his support and understanding, for always being there to share the joy and the hardness with me.

I would like to extend my gratitude to the SPA project's team members for allowing me to use their work in my research: Van Ho for the PDA e-mail interface, the User Interface Agent and the EMMA e-mail server; Alfred Krzywicki for the PDA calendar interface, the speech processor and the Calendar server; and Anna Wong for conducting the evaluation and designing the user questionnaire.

My thanks are also owed to my friends and colleagues at the School of Computer Science and Engineering for their support, encouragement and friendship: my three-year office-mate Liping Ma, Yi Luo, Woralak Kongdenfha, Alfred Krzywicki and Rita Chan.

I would like to thank Waleed Kadous for his help in setting up the IBM ViaVoice speech engine; Claude Sammut for supplying the ProBot software; Kee Siong Ng and Joshua Cole for supplying Alkemy and their help in resolving technical problems arising in the integration of Alkemy and the SPA; and Will Fitzgerald, Candy Sidner and Kristiina Jokinen for useful discussions about their work. I would also like to thank Agent Oriented Software Group for the use of their JACK agents platform.

Finally, I especially acknowledge the School of Computer Science and Engineering and the Australian Cooperative Research Centre for Smart Internet Technology for providing the financial support of my study.

It has been a long journey until the final words of this thesis and I owe much to many people for their contribution in this effort. Where no doubt I may have forgotten to name some of them, they deserve to be appreciatively acknowledged.

CONTENTS

1	Introduction	1
1.1	Conversational User Interfaces	3
1.2	The Smart Personal Assistant (SPA)	6
1.3	Practical Spoken Dialogues	9
1.4	Dialogue Adaptation	12
1.5	Overview of the Thesis	13
2	Dialogue Management	16
2.1	Linguistic Foundations	17
2.1.1	Speech Act Theory	17
2.1.2	Discourse Structure	20
2.1.3	Reference Resolution	22
2.2	Dialogue Management: An Overview	24
2.3	Approaches to Dialogue Modelling	27
2.3.1	State-Based Approaches	27
2.3.2	Frame-Based Approaches	29
2.3.3	Script-Based Approaches	33
2.3.4	Information State Approaches	34
2.3.5	Distributed Dialogue Models	38
2.3.6	Plan-Based Approaches	40
2.3.7	Agent-Based Approaches	43
2.4	Spoken Dialogue Management	45
2.4.1	Overcoming Speech Recognition Limitations	47
2.5	Multimodal Dialogue Management	49

2.6	Concluding Remarks	51
3	An Agent-Based Approach to Dialogue Management	52
3.1	Dialogue Characteristics	53
3.2	BDI Agents	59
3.2.1	The PRS Agent Architecture	60
3.3	The Dialogue Manager Agent	61
3.3.1	Conversational Acts	62
3.4	Plan-Based Dialogue Model	67
3.4.1	Discourse-Level Plans	71
3.4.2	Domain-Level Plans	76
3.4.3	Dialogue Model Reuse and Extensibility	78
3.5	Discussion	80
4	Dialogue Management in the Smart Personal Assistant	81
4.1	The Smart Personal Assistant Architecture	82
4.1.1	Overview	82
4.1.2	Speech Recognition	85
4.2	Dialogue Modelling	86
4.2.1	Syntactic Analysis	89
4.2.2	Domain-Specific Knowledge	91
4.2.3	User Model	95
4.2.4	Discourse Representation	96
4.3	Dialogue Processing	109
4.3.1	Domain Classification	110
4.3.2	Semantic Analysis	115
4.3.3	Intention Identification	117
4.3.4	Task Processing	131
4.3.5	Response Generation	133
4.4	Concluding Remarks	137

5	An Adaptive Agent-Based Dialogue Manager	139
5.1	Dialogue Adaptation	140
5.1.1	Existing Work in Spoken Dialogue Adaptation	142
5.1.2	Existing Work in BDI Agent Learning	143
5.1.3	Adaptation as Learning Plan Selection	145
5.2	An Adaptive Dialogue Agent	146
5.2.1	The Alkemy Learner	147
5.2.2	Adaptive Dialogue Responses	148
5.3	Summary	159
6	Evaluation	160
6.1	Evaluation of Spoken Dialogue Systems	160
6.2	Evaluation of the Smart Personal Assistant	163
6.2.1	Evaluation Setup	163
6.3	Evaluation Results	166
6.3.1	Objective Results	166
6.3.2	Subjective Results	189
6.4	Comments	191
6.5	Overall Remarks	193
7	Conclusion	195
7.1	Future Research	197
	Bibliography	199
A	Dialogue Management	211
A.1	Example ProBot Scripts	211
B	Dialogue Adaptation	214
B.1	Alkemy Learning Problem Specification	214
B.2	Alkemy Training Examples	219

C Evaluation	222
C.1 Training Tasks	222
C.2 User Questionnaire	222
C.3 Utterance-Level Performance for Each User	227

LIST OF FIGURES

1.1	A Sample Scenario with the SPA	4
1.2	Overview of the SPA System Architecture	7
2.1	Ranking of the Salient List Entities	23
2.2	Example State Diagram for a Book Service Application	28
2.3	Slot-filling frame for a movie show-time task	30
2.4	Example dialogue control rules used in Mercury system	31
2.5	The TrindiKit Architecture	35
3.1	Dialogue Illustrating Context Switching	55
3.2	Use of Back-End Information to Support Dialogue Management	56
3.3	PRS Architecture	61
3.4	Plan-Based Dialogue Model	69
3.5	Example Act Type Determination Plan	73
3.6	Example Intention Identification Plan	74
3.7	Example E-mail Determination Plan	77
3.8	Example E-mail Task Processing Plan	78
4.1	The SPA System Architecture	83
4.2	Example Scenario Illustrating Dialogue Management Process	87
4.3	Interaction of SPA Components	88
4.4	Example ProBot Scripting Rules	92
4.5	Example Verb-Task Mapping in Task Description	94
4.6	Discourse Segment Hierarchy	101
4.7	Example Discourse Stack	103

LIST OF FIGURES

4.8	References to Objects on the GUI	107
4.9	Dialogue Illustrating Reference Resolution Using the Salient List	109
4.10	Illustration of Dialogue Control Flow	111
4.11	JACK Plan Specification Schema	112
4.12	Dialogue Control for Domain Classification and Semantic Analysis	115
4.13	Semantic Structure Representation	117
4.14	Example Semantic Structure	117
4.15	Dialogue Control Flow for Intention Identification	118
4.16	Intention Identification JACK Plan	120
4.17	Dialogue Control Flow for Task Determination	122
4.18	Task Determination JACK Plan	124
4.19	Task Type Determination JACK Plan	125
4.20	Task Objects Determination JACK Plan	126
4.21	Temporal Phrase Resolution	129
4.22	Clarification Request Generation and Handling	130
4.23	Dialogue Control Flow for Task Processing and Handling	132
4.24	Return Message Content JACK Plan	134
4.25	Return Message Summary JACK Plan	134
4.26	GUI Output for E-Mail Displaying Task	135
4.27	GUI Output for Appointment Cancellation Task	136
4.28	GUI Output for E-Mail Deletion Task	137
5.1	Integrating Learning into the PRS Agent Architecture	145
5.2	Learning Plan Selection for Response Generation	150
5.3	Data Specification for Alkemy Learning Problem	152
5.4	The Predicate Rewrite System for Response Adaptation Problem	154
5.5	An Example Dialogue without Learning	155
5.6	Dialogue with Learning	156
5.7	Decision Tree Representing User's Preferences	157
6.1	Concept-Word Recognition Accuracy	168
6.2	Average Dialogue Length Per Task	169

LIST OF FIGURES

6.3	An Example Scenario for Task (6)	174
6.4	Example Failures of Task (10) and (11)	175
6.4	Example Failures of Task (10) and (11)	176
6.5	Source of Unexpected and Inappropriate Responses	180
6.6	Unexpected Response Caused by the User	181
6.7	Unexpected/Inappropriate Response Caused by the Partial Parser	182
6.8	Response Problem Caused by the Dialogue Manager (a)	184
6.9	Response Problem Caused by the Dialogue Manager (b)	185
6.10	Response Problem Caused by the Dialogue Manager (c)	186
6.11	Response Problem Caused by the Dialogue Manager (d)	187
6.12	Number of Clarification Requests Per User and Per Task	188

LIST OF TABLES

2.1	Types of Speech Acts	18
2.2	The Cooperative Principle	20
2.3	SUNDIAL Dialogue Strategies Example	32
3.1	Conversational Act Descriptions	64
3.2	Example Dialogue and the Associated Conversational Acts	66
4.1	Example Speech Recognition Output	86
4.2	Components of the Shallow Syntactic Frame	91
4.3	Example Syntactic Frame	91
4.4	Data Structure for Representing Domain Knowledge Description	93
4.5	Data Structure for Representing Conversational Acts	97
4.6	Conversational Acts and the Corresponding Act Contents	98
4.7	Data Structure for Representing Domain Task	99
4.8	Example Relation between Dialogue Segments	105
4.9	Expression Types of Discourse Entities	108
4.10	Example Heuristic Rules for Act Type Determination	121
4.11	GUI Action Data Type	127
4.12	Sample Discourse Segment's Stack Entries	133
6.1	Evaluation Tasks	165
6.2	Task-Level Evaluation Results	172
6.3	Utterance-Level Evaluation Measures	178
6.4	Dialogue Management Problem Distribution	183
6.5	Summary of the User Survey Ratings	190

LIST OF TABLES

6.6	Example Speech Recognition Output of <i>Yes/No</i>	191
C.1	User Survey Ratings	226
C.2	Performance at Utterance Level	227
C.2	Performance at Utterance Level	228
C.2	Performance at Utterance Level	229
C.2	Performance at Utterance Level	230
C.2	Performance at Utterance Level	231
C.2	Performance at Utterance Level	232
C.2	Performance at Utterance Level	233
C.3	Number of Clarification Requests	234

INTRODUCTION

The past decade has seen an enormous growth in the use of mobile devices such as PDAs and mobile phones. In the modern era, people are highly mobile and often out of the office; this type of go-anywhere device allows them to always be available for contact. In addition, mobile devices have introduced a new way of personal information management, which supports almost anytime, anywhere access to important information such as personal data, documents, the World Wide Web, etc. This is made possible by recent technological advances which allow these compact devices to be equipped with reasonably powerful processors, enough memory storage and adequate battery life. They are becoming increasingly useful and widely accepted, hence more and more applications are being developed, exploiting them to their fullest.

Recent innovations and advances in speech technology have enabled the use of the speech modality in human-machine interaction, which is necessary in many applications. For example, using speech interfaces is perhaps the most appropriate solution for blind users to interact with computer-based systems. Moreover, speech interfaces may be more feasible to be used on small computing devices because of the lack of screen space for a visible graphical interface. Modern home appliances are becoming more complex, thus having many more features, which cannot fit easily on a small graphical interface. Hence it may be more convenient for the users to interact with them through speech interfaces. In some computer-based applications such as aircraft control where timing is critical but manual interaction with computing systems is slow, speech interfaces

could also be more appropriate. Furthermore, given the increasing number of control functions being integrated in modern cars, there is a potential for the use of speech interfaces in vehicle systems so that users do not need to take their eyes away from the road or hands away from the steering wheel when using these functions. Finally, teaching and learning are based on communication, thus computer-aided learning/training systems would be more effective if they allowed speech-based interactions [Lewis, 1993].

In this research, we are interested in developing conversational speech interfaces for computer-based applications. Incorporating spoken dialogue as a major modality for user-system interaction is especially relevant in the context of mobile devices where traditional user interface peripherals may not be available or appropriate. The main objective is to develop the dialogue management component for such applications, enabling natural and coherent user-system interaction through a spoken natural language dialogue.

Mobile devices have been used to schedule activities, search information or maintain interaction with other people. The availability of high-speed wireless Internet access allows data to be retrieved and managed on the devices while being kept securely on remote servers. However, different task domains are currently handled by different applications, thus managing information from these various sources can be a difficult task. Hence we aim to provide a single interface to the unified set of different back-end applications, which allows the users to collectively manage their information. We have built a *Smart Personal Assistant* (SPA), a collection of personal assistants, each specializing in a specific domain. The user can remotely interact with the SPA to perform a variety of tasks such as scheduling and updating appointments, manipulating e-mail messages, etc. The SPA can collaborate with and support the user by hiding task complexity, such as combining information in several domains if necessary. Hence the user interface for the SPA must support effective input methods, enable easy switching between application domains, and be able to present data of different formats appropriately.

However, mobile devices do have limitations: their small keypad and stylus are not convenient for entering input and a typical screen is only capable of legibly displaying a few lines of text. Therefore there is an obvious potential for developing other sophisticated means of interaction (e.g. through spoken natural language dialogues) in addition to the conventional graphical user interface. There have existed systems with speech interfaces that focus on a command and control style of interaction. Nevertheless, command-based speech interfaces are just an adapted version of graphical control interfaces where normal commands or features are mapped into speech commands. Our research objective is to develop user interfaces that can allow more sophisticated interaction through spontaneous interactive natural language dialogue. Spoken dialogues are feasible in applications such as the SPA because although the tasks are complex, they are also constrained, so that most domain-related keywords are known, which helps overcome the limitations of speech recognition over large vocabularies. In Figure 1.1, an example is presented to illustrate the functioning of the SPA in a scenario where the user engages in a spoken dialogue with the system to remotely access his e-mail and calendar.

1.1 Conversational User Interfaces

The graphical user interface has been so far the main method for human-computer interaction, in which the user interacts with the computer through direct manipulation of graphical objects. In traditional computer systems running the DOS or Unix operating system, user-system interaction is done through text-based interfaces and in a function-oriented manner. The user types a command on the keyboard and the computer performs the corresponding task, e.g. executing a Unix command. With the invention of pointing devices, in particular the mouse, in the 60s, and windows in the late 80s, user interfaces have advanced to a higher level, in which the user operates on various objects such as

SPA Hello, how can I help you?

User Do I have any new e-mail?

SPA You have two new messages, one from John Lloyd about school meeting and one from Paul Song about travel policy.
Displays the two message headers.

User Show me the message from John.

SPA Here you are.
Correctly displays the message's content.

User Move it to the Meeting folder.

SPA Message has been archived.
Displays again the two message headers.

User Now show me my appointments for tomorrow.

SPA You have two appointments tomorrow, one with John and one with Anna.
Displays tomorrow's calendar showing the two appointments.

User Cancel the school meeting at 3 pm.

SPA Appointment has been deleted.
Displays tomorrow's calendar showing the other appointment.

User Thank you.

SPA You are welcome.

Figure 1.1: A Sample Scenario with the SPA

windows, icons or hierarchical menus, to perform tasks. This way of interaction is feature-oriented, relying on graphical control features of the graphical user interface.

However, research has been carried out since the 80s with the objective of developing conversational user interfaces to provide more natural user-system interaction. Graphical user interfaces are not always convenient to use. They require memorizing a set of pre-defined functions or operations, and tasks must be carried out as a sequence of these basic interaction steps. Thus, for some users, learning to use computers is not trivial. Complex tasks may take a long

time to accomplish because of the time required for performing an excessive number of steps and navigating the interface. For example, a typical task such as changing the font of an existing document might involve several steps: open the document using a suitable application; select the document text using either a key shortcut or the mouse; interact with the task-menu to choose a new font face and/or a new font size; and finally, save the document. To do such complex tasks, the user must be able to determine and carry out that sequence of actions. User interfaces should rather be result-oriented, focusing on what users want to do rather than on how to do it. In order to achieve this, conversational natural language dialogue would be an effective communication modality for interacting with computer-based applications, a valuable addition to the current GUIs. For instance, the above task could be accomplished easily using a natural language request: *“Change the font of this document to Arial twelve”*. However, a GUI is still necessary for displaying the document.

Conversational user interfaces can allow more flexible and effective interaction. Current trends have shown an increasing number of features in computer-based applications, which would require a more complex hierarchical graphical interface, making the interaction with the application even more difficult. In contrast, a conversational user interface is by nature more human-like. It allows the user to describe the tasks in a high level which does not require primitive steps to be explicitly formulated, thus even unfamiliar users would be able to use the system effectively. As in the previous example, natural language dialogue could allow any user to change the font of a document without knowing how to use the GUI of the available document processing software. Conversational user interfaces can also be useful in information retrieval domains such as question-answering, help-desk or search engine applications. They offer users more expressive power to specify their information requests than with keyword combinations. In addition, the conversational context can also be taken into account to incrementally refine the request and present the user with more relevant information.

Existing research on intelligent user interfaces suggests that user interfaces should exhibit to some degree intelligent agent-like behaviour. Initially, there were arguments over which of two interaction metaphors would be better for user interface development: the tool metaphor or the agent metaphor. In one view, the system interface is viewed as a tool that allows the user's direct manipulation of available options in order to perform some task, whilst researchers of the interface-as-agent view suggested that intelligent interfaces should better be conceived as agents. This view is strengthened by Chin's argument [1991] that an intelligent interface must exhibit to some degree autonomous and intelligent behaviour, such as taking the initiative to correct user misconceptions or suggest alternative actions. A more recent proposal by Horvitz [1999] was to improve user-system interaction by combining the tool and agent metaphors into a single framework. Thus for personal assistant applications on mobile devices such as the SPA, it is essential to provide this sort of mixed-initiative interface (e.g. through natural language dialogue) in addition to the conventional GUI due to the limitations (e.g. small keyboard, stylus for input, small screen for output) of mobile devices such as phones and PDAs.

1.2 The Smart Personal Assistant (SPA)

In this section, we present a brief introduction to the Smart Personal Assistant application (SPA) [Wobcke *et al.*, 2005], which sets the context for our research. The development of the SPA was motivated by the notion of a personal assistant promoted by Maes [1994] as a way to help users coping with information overload. The idea is a software agent that behaves as a personal assistant who collaborates with and supports the user in various ways such as hiding the complexity of difficult tasks, performing tasks on the user's behalf, and helping the user to manage his/her own activities or coordinated teamwork activities with other users. A set of specialist personal assistants can be integrated into a multi-agent system in which assistants cooperate with one another or are coor-

1.2 The Smart Personal Assistant (SPA)

minated by a coordinator agent [Azvine *et al.*, 2000]. The SPA is a collection of personal assistants, each specializing in a particular task domain. The users can interact with the SPA through a variety of devices such as desktop PCs, mobile phones or PDAs as with a single smart assistant, for manipulating their e-mail and calendar. The overall architecture of the SPA is shown in Figure 1.2. In this research, we are specifically interested in the dialogue management aspect of such applications, aiming to provide sophisticated interaction through spoken natural language dialogue.

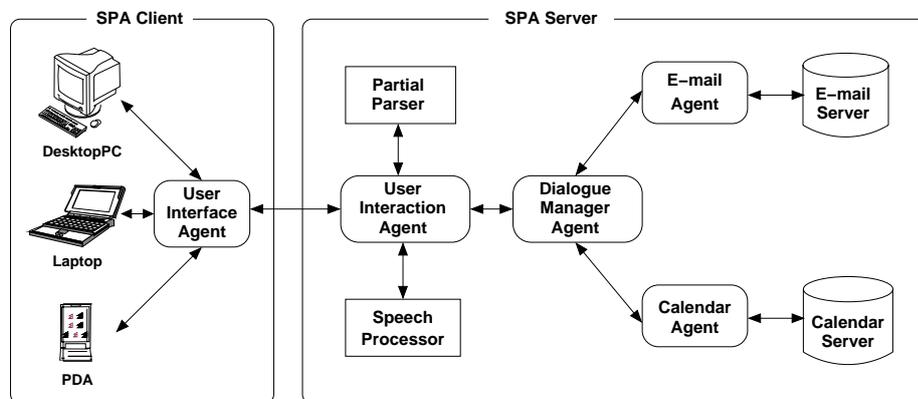


Figure 1.2: Overview of the SPA System Architecture

The SPA has been implemented as a multi-agent system using the JACK Intelligent Agents™ platform [Howden *et al.*, 2001], in which the agents communicate using message passing. There are two back-end task agents, which are wrappers built around the existing e-mail and calendar management software. The calendar system supports the usual tasks such as creating and modifying appointments. The email management software, EMMA [Ho *et al.*, 2003], enables the Dialogue Manager to request the system to perform tasks such as searching, deleting and archiving e-mail messages, and notifying the user on the arrival of important messages. In addition, EMMA can perform classification of e-mail in the Inbox into sorting folders and prioritize them based on the degree of importance, urgency, etc. This information is given to the user and also used by the Dialogue Manager in processing e-mail tasks.

1.2 The Smart Personal Assistant (SPA)

Although the SPA architecture allows the use of different user device types, the PDA has been chosen for system development to address the issue of mobility. User-system interaction, therefore, is through a simple kind of multimodal dialogue, including a speech interface and a graphical user interface on the PDA. The User Interface agent that resides on the user device monitors the user input and passes it to the User Interaction agent. Speech input is recognized by the Speech Processor, then processed by the Partial Parser to extract domain-specific keywords and other syntactic phrases. The Dialogue Manager uses the results to interpret the user's requests and delegates these tasks to the appropriate task assistants (agents). The system output is generated by the Dialogue Manager and returned to the user's device as speech and/or GUI output.

In order to support natural and flexible user-system interaction, developing the SPA requires consideration of both coordination and dialogue modelling aspects. Coordination of the specialist task assistants is required to provide a single point of contact for the user to maintain a coherent dialogue with the system while, possibly, switching the interaction to other task assistants or switching to use a different device. In addition, a sophisticated dialogue model is required to maintain the conversational context, allowing the system to interpret the user's actions on the interface and/or natural language utterances. Furthermore, the system may also need to exploit and combine information available in the back-end assistants in interpreting the user's intention.

To meet those requirements, dialogue management and coordination in the SPA application must be closely related. Thus it is necessary that the SPA have a central component, the Dialogue Manager, for handling both of these aspects. The research reported in this thesis is focused primarily on the development of the SPA's Dialogue Manager, which presents a single point of contact for the user to interact with the SPA, mediating communication between the user and the task assistants. Moreover, the Dialogue Manager must maintain the dialogue context and be able to recognize and fulfil the user's requests, possibly by delegating them to appropriate task assistants.

1.3 Practical Spoken Dialogues

With improvements in speech technology, numerous systems using spoken dialogue interfaces have been developed; most support *practical dialogues* that focus on specific tasks or goals such as planning travel trips, booking flights, seeking transportation timetables, etc. [Allen *et al.*, 2001]. Analysis of human-human and human-computer dialogues suggested that users still expect the system to retain many characteristics of a human service agent. McGlashan [1996] has proposed some of the required features as follows:

- *global structure*: most dialogues should have an opening, a body and a closing. This could also suggest that verification and acknowledgment be used for grounding the information being conveyed.
- *mixed-initiative*: both the user and the system should be able to take the initiative in the conversation. The information flow can be directed by both the user requesting the system or the system asking the user for missing information or clarification.
- *over-informativeness*: as often occurs in human-human dialogues, users may give redundant information that the system should be able to detect and ignore, or in some cases, take into account if the information is useful.
- *contextual interpretation*: user input may be partial or ambiguous, which can only be fully interpreted based on the current conversational context. Similarly, the user also interprets system responses as relative to the current context.
- *recovery from failure*: the system should have effective strategies for recovering from failures which can occur because of speech recognition errors or limitations of the system's internal knowledge. Users are easily frustrated by unsuccessful interaction attempts and excessive time spent on recovering from failures.

In recent years, considerable research effort has been directed at the area of dialogue management for computer-based applications. The goal is to provide high-level, human-like user-system interaction such as through conversational speech interfaces. With recent advances in speech technology, this has become feasible in domains that are sufficiently constrained to overcome the limitation of speech recognition over large vocabularies. Spoken dialogue systems allow the users to specify their requests and receive the results by conducting a natural language dialogue. Therefore, dialogue management in such applications is of particular importance to ensure effective but also flexible and adaptive interaction. The ultimate goal is to have the requests understood and accomplished successfully. However, the users should be able to express their requests in different ways, and modify or cancel them at any time. It is also desirable that the system's responses can be adapted to the conversational context, the user's physical context and preferences.

Our research goal is focused on the problem of managing (i.e. modelling and controlling) natural language dialogue for human-computer interaction, specifically, in applications for mobile devices. According to Churcher *et al.* [1997], the two main aspects to be considered in dialogue management are: firstly, mechanisms for maintaining the conversational context, and secondly, defining strategies to be adopted by the system in order to control the dialogue structure. An adequate model of the dialogue, including the conversational context, is not only necessary in interpreting the user's input but also helpful in predicting the user's next possible actions. The SPA's dialogue strategies should support mixed-initiative interaction, exhibiting reactivity in responding to the user's request and pro-activeness to allow recovery from failures.

Existing dialogue systems are mostly developed for some pre-defined task domain(s). The dialogue requirements are usually well specified and once the dialogue model has been developed, it is not intended to have further adaptation or extension for use in other applications. Hence the issues of reuse and extensibility have been given little consideration. However, being able to reuse a

dialogue model would reduce the cost of developing a dialogue system for a new task domain. Moreover, a multi-domain dialogue system would likely require the addition or removal of domain-specific back-end components, and the extension or modification of the dialogue model for handling different domains. This suggests a need to focus on the issue of extensibility in dialogue modelling. A generic dialogue model could be developed for a certain class of domains, which would enable the adaptation of the model for a new domain or its extension for handling additional tasks. Many existing dialogue systems make use of a dialogue model and a task model, however these are not always clearly distinguished. Our SPA application requires sophisticated dialogue interaction, and so requires a complex dialogue model, but not necessarily a complex task model. For our application, there is a potential that the system would be extended for integrating additional task assistants. Thus it is more important that the dialogue model can be reused, so that the system can be extended without much cost, at the same time maintaining the level of dialogue sophistication. Therefore, there should be a clear separation between domain-specific and domain-independent dialogue information and this requires the dialogue model to be at high level of modularity.

In this research, we propose an agent-based approach that employs a BDI agent architecture for developing the dialogue management component of spoken dialogue systems such as the SPA. The idea of employing an agent paradigm is motivated by the important agent-like requirements that the system must be reactive in fulfilling the user's request but also pro-active in recovering from errors or notifying the user of important events. Moreover, our approach is based on the theory of speech acts by Austin [1962] and Searle [1969]: the speaker, when speaking of an utterance is simultaneously performing three kinds of speech acts: *locutionary*, *illocutionary* and *perlocutionary* acts, in which the illocutionary force is the speaker's intention. Our motivation to employ an agent-based approach is further strengthened by earlier work on speech act theory and pragmatics of spoken dialogues. Perrault, Allen and Cohen [1978] view the coherence

of a dialogue as the contribution of its speech acts to the achievement of the speaker's goals. Similar to planning in problem solving, the speaker in the dialogue plans speech acts to satisfy a goal (e.g. affecting the hearer's beliefs, goals, etc.) [Cohen and Perrault, 1979; Allen, 1979]. Dialogue coherence, therefore, depends on the hearer's recognition of the speaker's plan. Making the hearer identify references in the speaker's utterance is also intended in the speaker's plan [Cohen, 1981]. We also consider dialogue acts as rational actions, exhibiting goal-directed behaviour. Using the BDI agent architecture allows the dialogue model to be explicitly represented in dialogue plans. Hence both dialogue and problem solving plans can be expressed in this common framework. This is required in the case of the SPA because of the close connection between dialogue management and coordination aspects.

We have developed the dialogue manager agent of the SPA by extending a BDI agent architecture, in which the dialogue model is encoded as a set of agent plans. These plans are at high level of abstraction, each specifying steps to achieve a goal corresponding to a dialogue aspect. In addition, discourse-level dialogue plans are separated from domain-level plans, thus enabling discourse-level plans to be reused in different SPA-type applications. The control flow of the dialogue is derived simply and automatically as the result of the BDI interpretation cycle, in which the most appropriate plans are selected according to the agent's beliefs and the current context.

1.4 Dialogue Adaptation

The usability of the SPA on a mobile device with small screen and limited bandwidth can be enhanced by adaptively tailoring responses according to the conversational context and the user's physical context, device and preferences. For example, e-mail messages are often too long to fit on one PDA screen, thus showing only a message summary could sometimes be more desirable. However, this solution may not be desirable for some users. Other users might prefer this behaviour but only if the e-mail message is not so important.

The user may interact with the system in different physical and social contexts: being at a noisy airport, in a one-speaker seminar or in a quiet office. This diversity in the device working environment also requires dialogue adaptation to provide appropriate interaction. For instance, spoken output should not be used in seminars or hospital wards because it would be socially inappropriate.

In this research, we also look at the problem of adaptive response generation for plan-based spoken dialogue systems such as used in the SPA, thus enabling the system responses to be tailored to the user's device, physical context and preferences. Because the SPA's dialogue manager uses plans for generating responses, the problem of dialogue adaptation becomes the problem of learning which dialogue plan to be selected and executed in a given context. We propose a general method of integrating learning into a BDI agent architecture, which allows the dialogue manager agent to learn plan selection, thus being able to provide dialogue adaptation.

1.5 Overview of the Thesis

The contribution of this research is twofold. Firstly, we consider the problem of dialogue management in spoken dialogue systems for personal assistants. We propose an agent-based approach for developing the dialogue manager by extending a BDI agent architecture. This BDI approach allows high level of abstraction in designing the dialogue model which consists of a set of modular plans, each associated with a communicative goal or dialogue aspect. There are other plans for handling domain-specific tasks. Thus this approach enables discourse-level plans to be reused in other applications.

Secondly, we consider the problem of adaptation of spoken dialogues in applications for mobile devices. In particular, we present a method for incorporating learning into a BDI agent architecture, allowing learning of the agent's plan selection, hence achieving adaptivity with plan-based dialogue management systems. This method enables the system to tailor its responses according to the conversational context and the user's physical context, devices and preferences.

The organization of the thesis is as follows:

- In Chapter 2, the existing work in linguistics that provides the foundation for our approach is presented, which includes the theory of speech acts and computational work in discourse and dialogue structure. Moreover, we give an extensive analysis of the existing approaches in dialogue management from simple methods such as *state-based*, *frame-based* and *script-based*; to methods based on *game theory*, *information state*; and more complex methods which are *plan-based* or *agent-based*. We also consider in this chapter specific issues for developing spoken and multimodal dialogue systems.
- Chapter 3 gives an overview of our agent-based approach to dialogue management [Nguyen and Wobcke, 2005]. First, we discuss the dialogue requirements of the SPA application and the advantages and disadvantages of various approaches and why they are not suitable for such applications. We then present our motivation to employ an agent-based approach. We argue that dialogue management in the SPA requires many agent-like characteristics, which motivates the use of a BDI agent architecture for developing the SPA's dialogue manager agent. The dialogue model, which is explicitly represented as (agent) dialogue plans for handling different discourse-level and domain-level aspects, will be described in detail. The modularity of this approach allows the dialogue model to be extended and reused in other applications [Nguyen and Wobcke, 2006b].
- Chapter 4 describes the dialogue management process in the SPA in detail. We first describe the SPA overall architecture and its various components, including the user interface, the speech processor, the syntactic parser, the Dialogue Manager and the back-end e-mail and calendar task assistants [Wobcke *et al.*, 2005]. The system is implemented as a multi-agent system using the JACK agent development platform, in which the Dialogue Manager is also a JACK agent. Using our agent-based approach,

dialogue processing is done automatically as the result of the Dialogue Manager agent's plan selection by the BDI interpreter. Dialogue examples will be given to illustrate how dialogue plans get selected and the dialogue management process is performed, which can be roughly divided into four sub-processes: analysing the syntax and semantics of the user's input, recognizing the user's intention, processing the requested tasks and generating and returning the output to the user [Nguyen and Wobcke, 2005].

- In Chapter 5, we address the problem of achieving dialogue adaptation in the SPA application, which can be considered the problem of learning plan selection strategies in our plan-based dialogue model. We propose a general method for incorporating learning in a BDI agent architecture, which can be used to achieve adaptation in plan-based spoken dialogue systems [Nguyen and Wobcke, 2006a].
- Chapter 6 reports the evaluation results of the SPA. Firstly, we review two existing methods for evaluating dialogue systems and discuss whether they would be suitable for our evaluation purposes. We then describe our evaluation setup and present and discuss the objective and subjective results. Overall, a high task completion rate was achieved, which indicates the potential of our agent-based approach in developing spoken dialogue systems.
- Finally, we give our conclusion and discuss the future work in Chapter 7.

DIALOGUE MANAGEMENT

Conversational dialogue is a kind of discourse, therefore, similar to other kinds of discourse, is an instance of the use of language. A dialogue is a cohesive linguistic unit which is composed of a sequence of natural language utterances, carrying a conversation of two or more persons. For example, unlike other kinds of discourse such as monologues, a spoken dialogue has at least one speaker and one hearer who communicate on some common topics. User-system interaction in spoken dialogue systems exhibits similar characteristics to those of human-human dialogues. Therefore computer-based management of conversational dialogues requires the use of techniques in related areas such as natural language processing (e.g. methods for syntactic and semantic analysis of sentences); discourse theory (e.g. how utterances in a discourse are coherently related and how past utterances can be used in interpreting the upcoming ones) and pragmatic analysis of dialogue such as the theory of speech acts (i.e. the idea that people speaking language are performing some kinds of communicative acts in order to accomplish their goals).

In this chapter, we first introduce the linguistic foundations of our research, which are common to many frameworks for dialogue management: the theory of speech acts and other important theories in discourse analysis. We shall then discuss the general problem of dialogue management and analyse the applicability of existing approaches in personal assistant applications. Our application, the Smart Personal Assistant, which provides both natural language and GUI access to e-mail and calendar, requires only limited support for multimodal input and output. Thus we shall briefly consider some important issues in multimodal

dialogue management. We also discuss the specific requirements for developing spoken dialogue applications, such as dialogue strategies for recovery from speech recognition errors.

2.1 Linguistic Foundations

2.1.1 Speech Act Theory

The theory of speech acts originated in the field of the philosophy of language and was first proposed by Austin [1962], in which he points out the distinction between *performative* and *constative* utterances. He argues that not all utterances are statements describing facts that must be either true or false (i.e. constative utterances) but there is another type of utterances, i.e. performatives, which are not only used to describe an action but to actually perform it. Moreover, none of these performative utterances are either true or false; they are rather *felicitous* or *infelicitous*. Hence constatives are for “saying” things while performatives are for “doing” things such as in the utterance: “*I appoint John to be the executive manager*”, which is felicitous if the speaker had the right to make the appointment and the effect of the performance of the utterance is John successfully becoming the executive manager.

However, Austin did not give absolute criteria for distinguishing these two kinds of utterances, perhaps because an utterance can be either a constative or a performative, depending on the context in which it has been uttered. For example, “*I take the blue cap*” may have a performative force of making an order or it may be just making a statement. Not only the context but the use of special features of spoken language such as the tone of voice could make the utterance a performative.

Searle [1969] has proposed a more comprehensive version of the theory of speech acts, which does not distinguish constatives from performatives but considers constatives as also the performance of a speech act, e.g. the illocutionary

act of stating a fact. According to Searle, speaking a language is engaging in a rule-governed form of behaviour to perform speech acts. Moreover, uttering a sentence is simultaneously performing three kinds of speech act: *locutionary* (a locutionary act contains an utterance act and a propositional act), *illocutionary* and *perlocutionary* acts, according to different aspects of the speaker's intention. Table 2.1 gives a description of these types of speech acts.

Act Type	Description
<i>c</i> <i>a</i>	the act of uttering a string of words, an utterance, in a language.
<i>c</i> <i>a</i>	the act of stating, questioning, commanding, etc.
<i>e</i> <i>c</i> <i>a</i>	the act representing the notion of consequences or effects of the three acts above have on the hearer. For example, by making a statement the speaker may make the hearer convinced, by making a request the speaker may get the hearer to do something, etc.

Table 2.1: Types of Speech Acts

Different utterances may have the same reference (to a certain object) and predication, yet have different illocutionary forces, such as in the following example (adopted from [Searle, 1969, p.22]). The four utterances are respectively a statement, a question, a command and an expression of desire even though they have the same propositional content.

1. *Sam smokes habitually.*
2. *Does Sam smoke habitually?*
3. *Sam, smoke habitually!*
4. *Would that Sam smoked habitually.*

In the speech act theory, the unit of language communication is not linguistic elements such as words, sentences, etc., but rather the production of utterances in performing a speech act. Moreover, speech acts are made and performed in accordance with certain rules by the use of linguistic elements. For an utterance, the order of words, the stress, the main verb, the verb mood, etc., can serve as the indicating devices for recognizing the underlying illocutionary force.

According to Grice [1957], when uttering an utterance, the speaker intends to induce a belief in the hearer, and also intend that the utterance is recognized as so intended. Thus the coherence of a dialogue depends on the hearer's understanding of the speaker's intentions, and in order to identify the speaker's intentions, it is useful to recognize the associated speech acts or the illocutionary force of the utterance being performed, Bruce and Newman [1978], Allen [1979] and Cohen and Perrault [1979]. Likewise, in spoken dialogue applications, the system needs to identify the user's intentions to be able to fulfil their requests. The theory of speech acts, therefore, has been used as a common linguistic foundation in the development of many existing dialogue systems, which helps realizing the requirements of modelling and processing dialogue: the recognition of the propositional acts, e.g. the objects that are referred to, the predications on these objects, and importantly, the illocutionary acts being performed and so the user's intentions.

Moreover, Searle has pointed out that, because the basic unit of linguistic communication is the production of an utterance in the performance of a speech act, the action of making a noise or a mark on a piece of paper can also be considered an instance of linguistic communication if it is done by oneself and with certain kinds of intentions. Therefore, in multimodal dialogues, the user's actions in modalities other than speech can be considered a part of the intentional communication and hence modelled using speech acts.

Although the theory of speech acts indicates the existence of perlocutionary acts, i.e. the effects or consequences for the hearer by the speaker in performing illocutionary acts, the practical use of the theory in dialogue applications does not require modelling of these acts because in most cases, the perlocutionary effect of the utterance is closely related to its illocutionary act. The reason is that the user, by nature, typically has a cooperative attitude towards the system, i.e. having requests understood and carried out successfully, as formulated in the *Cooperative Principle* by Grice [1975]. Similarly, the system can be developed to be cooperative as well. Therefore, in many human-computer dialogue systems,

we can assume that each illocutionary act is to communicate only one intention and this intention is made as clear as possible to be recognized. The four maxims of the *Cooperative Principle* and their explanation are given in Table 2.2.

Maxim	Explanation
QUANTITY	make your contribution as informative as required but not over-informative
QUALITY	do not say anything you believe as false or lack adequate evidence
RELATION	be relevant
MANNER	be brief, orderly and avoid obscurity, ambiguity

Table 2.2: The Cooperative Principle

The speech act theory of Austin and Searle provides the theoretical basis for understanding that language is not just a means of communicating information but also a mode of action, i.e. speaking language is performing communicative acts according to different aspects of the speaker’s intentions. In personal assistant applications, the user, when speaking an utterance, has the intention of having some task(s) done, and this intention is intended to be recognized by the system. The theory of speech acts provides important concepts for modelling the communicative acts performed by dialogue participants, thus has application to dialogue modelling in personal assistant applications.

2.1.2 Discourse Structure

Dialogue is a kind of discourse where each participant periodically takes turns to be the speaker and the hearer. Dialogue management requires understanding of the structure of a dialogue and the notion of a coherent dialogue. Semantically, each utterance in a coherent dialogue must somehow relate to each other utterance of the dialogue so that interpretation and generation of linguistic phenomena such as anaphoric references or temporal expressions is possible.

Discourse research has been closely related to a question of what information is being conveyed in a coherent discourse, in addition to the literal meanings of the individual utterances. There are two major lines of approach which are referred to as *informational* and *intentional*. Informational approaches consider the coherence of discourse to derive from semantic relationships between the information expressed by successive utterances [Mann and Thompson, 1988]. On the other hand, according to intentional approaches, the coherence of discourse comes from the intentions of the speaker, and understanding depends on the hearer's recognition of those intentions.

Among the intentional approaches, computational work in discourse by Grosz and Sidner [1986] has identified three components of discourse structure: the linguistic structure (i.e. the sequence of utterances), the intentional structure and the state of focus of attention. Utterances in a discourse naturally form a hierarchy of discourse segments, which is the discourse's *linguistic structure*. In addition, each discourse segment has a *discourse segment purpose* or *discourse segment intention*. The *intentional structure* of the discourse contains these segment intentions and also captures the relations between them. Discourse intentions differ from other kinds of intentions in that they are intended to be recognized. Nevertheless, the speaker, when communicating an utterance, may often have other intentions that are intentionally hidden from the hearer.

According to Grosz, Weinstein and Joshi [1995], the *attentional state* models the discourse participant's focus of attention at any given point in the discourse. It has two components for modelling the changes of attention within the local discourse segment and at the global level, which correspond to the coherence of the discourse at the local and global level. Coherence of the dialogue at the global level is affected by the intentional structure, i.e. the relationships among the discourse segment purposes. The local coherence among segment utterances, on the other hand, depends on the compatibility between the utterances' centring objects and their subsequent referring expressions. In addition, the attentional state has an important part which contains the salient entities that have been

mentioned earlier in the discourse. Thus references to these entities could be resolved by searching for matched objects that are currently in attention.

Dialogue modelling for personal assistants requires a computational model of discourse for representing and accessing information about the conversational context. The intentional approach of Grosz and Sidner is a suitable framework, which is based on recognizing and using intentions to represent the discourse structure, while in conversational dialogues, the intentions of the dialogue's participants can be considered the discourse intentions. In addition, this approach is consistent with the earlier work on speech act theory.

2.1.3 Reference Resolution

Reference is a common linguistic phenomenon, in which a linguistic expression is used to refer to some discourse entity, i.e. its referent. The referents can be entities in the real world such as a person, an object or the time of an event occurrence. In conversational dialogue, the use of references is very common. Moreover, it is necessary that the hearer can resolve these references in order to interpret the speaker's utterance, thus recognize his/her intention. Anaphora is the most common type of reference, which is the use of certain words to refer to other referring expressions mentioned previously in other utterances or together in the same utterance. For example, a previously mentioned noun or noun phrase can be referred to using a pronoun or a definite noun phrase as in the following sentences:

Mary was on her way to work when *she* saw the accident.

I knew *Mary* when I was in university. *That girl* is super smart.

Grosz, Weinstein and Joshi [1995] have claimed that certain discourse entities mentioned in an utterance are more central than others. Inspired by this centring theory, Strube [1998] proposes a computational model that can be used to resolve anaphoric references called the *salient list*. The salient list contains

some discourse entities that are currently in focus for the hearer, and their rank order (based on their degree of salience) provides some indication of how likely they will be referenced in future utterances. The concept of salience originates from cognitive psychology, and refers to the degree to which a particular object stands out relative to other objects in a situation. In Strube’s computational model, the salience of discourse entities are based on their information status or expression type. The rank order of the salient list provides preferences for interpreting anaphoric references. There are three types of expression for representing discourse entities, as follows, and the ranking criterion is shown in Figure 2.1 (adopted from [Strube, 1998, p.1253]).

- OLD: Refer to old discourse entities that have been mentioned previously in the discourse. This type of expression is further divided into *evoked* and *unused* entities.
- MED: Refer to mediated discourse entities and consists of three different types: *inferables*, *containing inferables* and *anchored brand-new*.
- NEW: Introduce *brand-new* discourse entities, which are mentioned in the discourse the first time.



Figure 2.1: Ranking of the Salient List Entities

Anaphora resolution can be done by searching the salient list from left to right according to the rank order. The objects in the referring expression are checked against discourse entities in the salient list until a matched type has been found. After that, the salient list must be updated accordingly, which may require re-positioning some elements of the list to conform with the ranking criteria. In

addition, brand-new objects mentioned in the dialogue are immediately inserted into the salient list at their determined positions.

For dialogue applications such as e-mail and calendar management, it is desirable that the user be able to use references such as anaphora and temporal expressions. For example, the user might use pronouns to refer to people, e-mail messages or topics. Temporal expressions might also be used frequently for making and changing appointments. To provide sophisticated interaction, the system must be able to perform reference resolution correctly and effectively. Therefore, the use of a computational model such as the salient list for maintaining discourse entities is required, which can be used not only for resolving references but also generating context-sensitive natural language responses.

2.2 Dialogue Management: An Overview

In recent years, considerable research effort has been directed at the area of dialogue management for computer-based applications. The goal is to provide high-level, human-like user-system interaction such as through conversational speech interfaces. Most existing dialogue systems have a special component, the dialogue manager, for handling the dialogue interaction. However, the dialogue model can be distributed in several system components and dialogue management performed by more than one system component. The main functions of the dialogue management component are maintaining the conversational context and having effective strategies for controlling the dialogue flow. The dialogue model must specify how the dialogue evolves from one state to another in response to the user's current utterance and according to the conversational context. Whether being centralized or distributed, the dialogue model must include up-to-date information about the current state of the dialogue. Moreover, it is likely that some domain-specific knowledge is required in interpreting the user's utterances, handling the user's requests and generating suitable responses. Thus the dialogue model should also contain necessary domain knowledge.

Existing approaches to dialogue management use different methods for modelling the dialogue's informational components and control strategies. Systems that are required to handle only simple and constrained tasks often employ simple but effective approaches, such as using a finite state machine to represent all possible dialogue states. In addition, dialogue control strategies, i.e. the state transition model, are either pre-specified by the system developers or learned from training examples. In contrast, there are more sophisticated methods, in which dialogue is viewed as a collaborative problem solving process. Systems that employ these approaches can handle more complex and less constrained tasks. Dialogue management in most of these applications involves complex plan recognition, as for problem solving.

Depending on the required complexity of the dialogue interaction, an adequate dialogue model is needed to provide coherent interaction between the user and the system. In command-based speech applications, each user's utterance corresponds to a single task, thus the system can perform effectively by just responding to the user's requests in a reactive manner. Nevertheless, sophisticated dialogue systems must support requests that require user-system interaction spread over multiple utterances and multiple turns such as to provide more information or clarification. Therefore, the dialogue model must allow for the system to maintain the conversational context and to be pro-active in requesting additional information, recovering from errors or notifying the user of important events.

In the following section, we present existing approaches to dialogue management in the literature, ranging from simple but effective to more complex methods. We classify these approaches into different categories according to how they handle the two main aspects of dialogue modelling: the representation of the dialogue's information state and the specification of dialogue control strategies. In multi-domain applications such as the SPA system, there is a potential to reuse and/or extend the dialogue model for handling different task domains, thus we shall also discuss how the existing methods support the issues

of reuse and extensibility. Among the existing approaches, perhaps the frame-based method has been the most popular, which uses frames of filled/unfilled slot-filling values for the underlying representation of the information state and models dialogue interaction as a set of pre-defined rules for filling these values. In addition, there have been other methods that employ state-based graphs or scripts for dialogue modelling. These approaches have been shown to be effective in handling simple and constrained tasks. Their main weakness is that they are limited in supporting complex mixed-initiative dialogue interactions and allowing complex task models. Moreover, the maintenance and extension of the state and transition sets, the topic hierarchies and/or rule sets requires significant effort and the dialogue models generally do not scale well as the dialogue interaction becomes more complex.

Another method is based on the concepts of information state and dialogue moves. The information state represents the relevant aspects of information in the dialogue and is updated in every dialogue move according to a set of update rules. This approach allow modelling simple goal-directed discourse behaviour through the use of dialogue moves. Although complex models of the information state can be achieved, there still is the problem of defining and maintaining a large set of update rules, which makes it difficult to control complex dialogue interaction.

There has been other work on modelling dialogue for complex task domains, which views dialogue as a collaborative process in which the user and the system work together for some problem solving. Plan-based approaches are based on using plans to model the problem solving tasks, considering dialogue interaction the process of task planning and execution. As an extension of the plan-based method, agent-based approaches view dialogue interaction as rational action, exhibiting goal-directed behaviour. Using an agent-based approach, the dialogue structure automatically emerges according to the agent's rationality principles. It is possible to model common discourse phenomena such as grounding, sub-dialogues for confirmation, clarification, negotiation, etc. An agent-based

method can allow robust and sophisticated mixed-initiative dialogue interaction. We now consider each approach in more detail.

2.3 Approaches to Dialogue Modelling

2.3.1 State-Based Approaches

The main idea behind state-based methods is to enumerate all possible dialogue states and allowable transitions between them. The dialogue model is explicitly represented using a finite state machine or graph of pre-defined conversation states, in which each state is defined by a set of state variables representing different aspects of the ongoing conversation. For each state, there can be a number of possible transition choices, leading to the next state. The transitions between states are determined by the system's dialogue strategies. Figure 2.2 shows the state diagram for a telephone-based book service [Larsen and Baekgaard, 1994]. At each state, the system prompts the user for choosing one of the available options or providing more information, which will lead to a next state. The number of states needed for this simple book service is 11, which could grow and become unmanageable as the problem becomes more complex.

In the Syrinx spoken language system [Estival, 2002], a state-based dialogue model for each domain is defined using a scripting language, which specifies possible paths through a course of interaction and the system's actions to be carried out at each dialogue state. The dialogue states and associated system actions are designed by the system developer and interpreted at runtime by the dialogue flow controller. Dialogue strategies for systems using state-based approaches can be pre-specified during design phase, e.g. the CSLU toolkit [Novick and Sutton, 1996]. However, dialogue strategies can also be learned from a corpus of training examples. The Email Voice Interactive System (ELVIS) [Walker *et al.*, 1998; Walker, 2000] is a speech application for the e-mail management domain which employs reinforcement learning to learn the

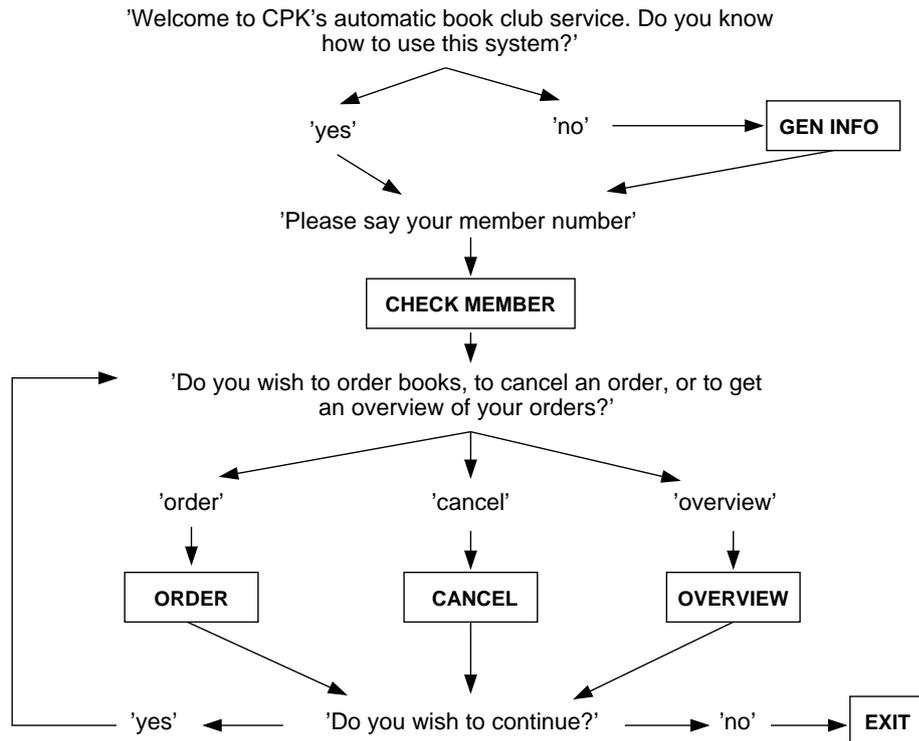


Figure 2.2: Example State Diagram for a Book Service Application

strategy choice selection. ELVIS allows access to e-mail over fixed-line phone in a simple question-answer fashion. In this system, for each strategy type (e.g. provide-info, summarization task, etc.), there are a number of possible strategy choices leading to different state transitions. For example, to give the user information about the current state of the Inbox, the system can choose to give a brief summary, a summary and the subject of the newest message, or a choice-prompt for selecting which message to be shown.

State-based approaches are useful for simple applications with strict system-initiative. An advantage is that as the system takes initiative all the time and the user is given only a few choices in response to a system question, it can provide useful constraints for improving speech recognition performance. State-based approaches, however, are not practical for complex domains because a typical conversation would require a large number of states. Moreover, a sophisticated dialogue that supports mixed-initiative would require very complex dialogue

strategies, resulting in a complex transition model. In addition, if the system supports more than one topic, the user cannot switch back and forth between topics in the middle of an ongoing conversation. Furthermore, it is not always possible for the user to go back and correct information supplied earlier. It is even impossible for the user to provide over-informative answers as the system would ignore the additional information. In more complex applications, several mixed-initiative dialogue interactions are often required to complete a single task. Designing a state-based dialogue model is only computationally effective for a small number of states, hence state-based methods are more suitable for developing command-based speech interfaces than for modelling sophisticated dialogue systems. Finally, state-based dialogue models contain largely domain-dependent information, i.e. each state is defined by a set of variables representing different aspects (most are domain specific) of the ongoing conversation. Thus this method does not support reuse; adapting a state-based dialogue system for a new application domain requires developing a new set of dialogue states and transitions.

2.3.2 Frame-Based Approaches

Perhaps the most popular approach to dialogue management is the frame-based method. Although in practice, existing systems have employed different variations, they share a general framework, in which dialogue interaction is considered a slot-filling problem. The dialogue model typically includes a hierarchy of frames as the representation of the information state. Each frame is used for encoding a task-related collection of information, often a set of filled/unfilled variable values (slot-filling values). Thus the role of these systems is to find the appropriate values for the required variables, then create and perform database queries to retrieve the user's requested information or execute other operations to fulfil the user's request. The frame-based method may also be referred to as a "rule-based" method because dialogue control is governed by a pre-defined set of ordered rules which specify what actions to take when a variable has or

2.3 Approaches to Dialogue Modelling

has not been filled. In other words, these dialogue control rules specify the system's dialogue strategies. An advantage of this method, in comparison to the state-based method, is that systems using frame-based approaches can support mixed-initiative dialogue. Moreover, the user can actively provide information and provide more information than required.

Frame-based approaches can support more flexible dialogue interaction than state-based approaches because dialogue flow does not have to strictly follow the pre-defined state transitions. However, the frame-based method is still limited because it can only allow a simple task model that contains a fixed set of required variables. Therefore, most frame-based dialogue applications are focused on simple and constrained tasks such as database information retrieval [Jönsson, 1997]; providing information over the phone about train timetables (the Philips system [Aust *et al.*, 1995]), ferry information (Waxholm [Carlson, 1996]) or movie show-times (MIMIC [Chu-Carroll, 2000]). The Philips system uses a dialogue description language (HDDL) to specify dialogue sections (frames), the associated variables and dialogue rules for extracting variable values. Similarly, in the Waxholm system, each dialogue topic, e.g. getting a timetable or showing the map of a place of interest, is represented as a frame with associated slots. The dialogue topic is identified by examining the dialogue history and considering the existence of slot values. Figure 2.3 shows a frame used in MIMIC for specifying the required variables of the movie show-time request task.

Question-Type:	When
Movie:	<i>mandatory</i>
Theater:	<i>mandatory</i>
Town:	<i>optional</i>

Figure 2.3: Slot-filling frame for a movie show-time task

More recently, several travel planning dialogue systems have been developed within the DARPA Communicator Project, most employ the frame-based method, such as the University of Colorado's CU dialogue system for travel and

2.3 Approaches to Dialogue Modelling

information state over different levels: linguistically-oriented and task-oriented frames. The interesting point in this work, however, is how feedback to the user is affected by dialogue degradation and recovery according to the dialogue strategies. For example, as a dialogue degrades, i.e. the system has repeatedly failed to interpret the user input, its confirmation strategy changes from being implicit to explicit, and finally, at the lowest level, is replaced by a request-spell strategy (an example dialogue is shown in Table 2.3).

Conversation		Dialogue Strategy
User	Leave London at 3 pm.	. . . c c f i a
System	<failed to recognize> Where and when will you leave?	
User	London at 3 pm.	e c . . . e - a a e e c f i a
System	<failed to recognize> Where will you leave?	
User	London.	e e - e
System	<failed to recognize> Could you spell that?	

Table 2.3: SUNDIAL Dialogue Strategies Example

Frame-based approaches rely heavily on using domain-specific information in the linguistic structure of the current and previous utterances and simple heuristics for filling the values of required variables and matching dialogue control rules. The system then takes the actions specified in the matched rules to perform back-end database access or other dialogue actions such as requesting the user for more information. The major disadvantage of the frame-based approaches is that design and maintenance of the system's frame hierarchy and/or rule set requires significant effort. There is also the issue of achieving modularity with this method. Appropriate priorities of the control rules are important but hard to specify. If the dialogue processing fails, it would be difficult to identify the problem and the associated rules. Developing the rule set for complex task

domains usually requires collecting and analysing a large number of training dialogues. In frame-based approaches, although some of the rules or frames can be reused, extending a frame-based dialogue model for handling additional task domains is difficult because of the lack of modularity. Hence a dialogue model with higher level of abstraction and modularity is needed to enable the reuse and extension of the dialogue across different application domains.

2.3.3 Script-Based Approaches

The ELIZA computer program [Weizenbaum, 1966] is perhaps the first “dialogue system”, a conversation simulator implementing a Rogerian psychologist. ELIZA is better described as a script-based chat-bot that is able to chat with humans in typed natural language, however it cannot understand and perform tasks. A ELIZA script contains rules used to look for certain patterns of words in the user’s input, and reply with a pre-determined output. Although ELIZA does not have a real dialogue model, it has worked effectively as a chat-bot because people themselves tend to attribute meanings to their past experience that were not intended by ELIZA.

More recently, there has been work that attempts to facilitate the problem of developing and maintaining dialogue control rules by grouping them into contexts (scripts), each corresponding to some dialogue topic, e.g. ProBot [Sammut, 2001]. In ProBot, each rule contains a pattern and a response, where the pattern is used to match with some domain-specific features of the user’s utterances. Thus in practice, constructing and prioritizing these dialogue rules is time consuming. Although some degree of modularity can be attained with this script-based method, still there is the basic problem of maintaining a large rule-base of scripts. This makes the control rules in a scripting system difficult to specify and hence scripting systems do not tend to scale well. There is also the difficulty of reusing and extending these scripts.

2.3.4 Information State Approaches

The information state approach [Traum and Larsson, 2003] views dialogue interaction as a sequence of information state updates, which is the result of the performance of dialogue moves. In this approach, dialogue modelling requires consideration of several components as follows:

- Conceptual descriptions and formal representations of the informational components. The informational components capture different aspects of the dialogue, which include static information such as domain knowledge or dynamic information such as the conversational context, dialogue participants, their obligations, commitments, etc.
- A set of dialogue moves that triggers the update of the information state. Language interpretation rules are required for recognizing the associated dialogue moves from the speaker's utterances.
- A set of update rules which governs the updates of the information state given the current status of the information state and the recognized dialogue moves. An update rule contains a list of preconditions, which must be true on the information state for the rule to be applicable. The effect list of the update rule defines operations to be applied to the information state.
- An update strategy for determining, from the set of applicable update rules, which rule(s) to apply. The simplest strategy is to select the first applicable rule.

Based on the notions of information state and dialogue move, the TRINDI Dialogue Move Engine Toolkit, the TrindiKit [Larsson and Traum, 2000], provides support for developing information state-based dialogue systems, allowing making choice of the representations and implementations of the informational components, dialogue moves, update rules and update strategies. The TrindiKit

architecture is presented in Figure 2.5, in which the dialogue move engine updates the information state according to the selected update rules on the basis of recognized dialogue moves. DIPPER [Bos *et al.*, 2003] is a another multi-agent architecture for prototyping spoken dialogue systems, which is implemented on top of the Open Agent Architecture (OAA) [Martin *et al.*, 1999], and has its dialogue management component based on the information state approach. Other components such as for speech input and output, natural language understanding and generation, etc., are software agents built on existing off-the-shelf software. In comparison with TrindiKit, the main difference in DIPPER is the use of a different language for defining information update and dialogue move selection rules. In addition, the multi-agent architecture of DIPPER allows replacing system components with others of the same functionality.

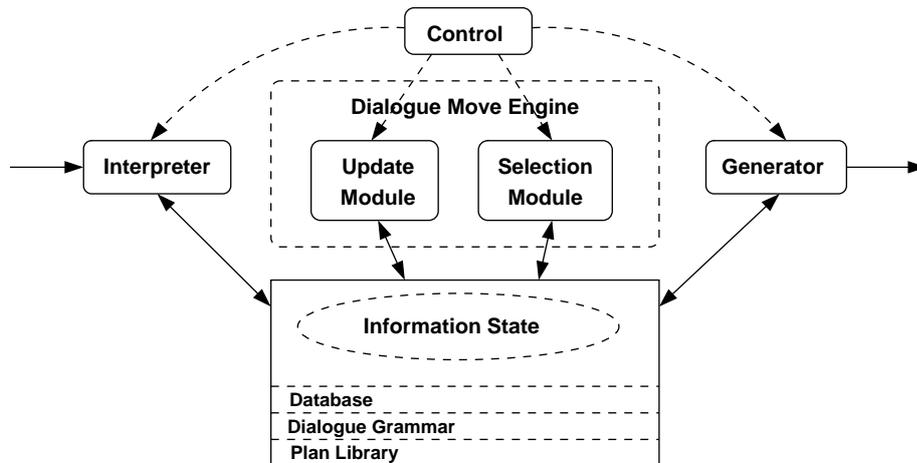


Figure 2.5: The TrindiKit Architecture

The TrindiKit architecture has been used to develop dialogue systems such as EDIS [Matheson *et al.*, 2000] and MIDAS [Bos and Gabsdil, 2000] for route information service and GoDIS [Bohlin *et al.*, 1999] in the travel agency domain. However, neither of these systems has been fully implemented according to the proposed design specification and they handle only simple, mainly system-initiative, dialogue interaction.

In the GoDIS system, the most important components of the information state include a list of actions that the system wants to carry out (i.e. a plan corresponding to the recognized task), the short-term actions that the system is going to do next (e.g. what question to ask the user) and a stack of questions under discussion. When the system asks a question, it is pushed onto the top of the stack; when the question is answered, it is popped off the stack. This means every answer of the user must correspond to a question under discussion. The research focus was on question and task accommodation in handling elliptical utterances. An elliptical utterance is always assumed to be of a `reply` move, which will trigger a special update rule for determining a matching question from the current plan. There is another update rule for accommodating tasks if the system has no plan yet in the information state. Dialogue control in GoDIS is similar to that of the frame-based dialogue systems, considering the system's plan as the current frame and searching for the relevant question under discussion filling in some slot value(s).

In contrast, the emphasis in the EDIS system is on realizing social commitments and obligations of the dialogue's participants as the effects of the dialogue acts (moves) performed. Obligations are actions that the system is obliged to perform while commitments are the system's accepted propositions. Thus the information state contains grounded and ungrounded information, i.e. obligations and commitments. Ungrounded information becomes grounded as a result of a grounding act, the `ack` move.

The MIDAS system uses discourse representation structures proposed in the Discourse Representation Theory (DRT) of Kamp [1981] as the major component of the information state. An utterance is analysed and transformed into discourse representations structures, which serves as the context for processing the next utterance. A discourse representation structure contains a discourse referent and its associated conditions, represented in a first-order logic. MIDAS uses theorem provers for recognizing dialogue moves and performing information state updates.

2.3 Approaches to Dialogue Modelling

There has been other work using the information state approach, based on Conversational Games Theory [Power, 1979], which models dialogue moves as moves exchanged in conversational games. This approach considers the user participating in a task-oriented dialogue to engage in a series of conversational games, each corresponding to a task request. Moreover, each utterance in the dialogue can constitute one or more moves in its relevant game. For instance, the MailSec system of Williams [1996], a telephone-based application for remote access to e-mail, uses six types of games and twelve types of moves. A game typically consists of just a pair of moves but can also be nested. The system maintains a stack of games and moves as the core dialogue data structure, its information state. Dialogue control in MailSec is simple, using simple syntactic features to recognize a single move from the user's utterance. Because most game moves go in pairs, the selection of the system's moves and so the update of the dialogue data structure is straightforward.

Based also on the notion of conversational games, dialogue control in the SRI Autoroute system [Lewin, 1998] is more complex than that of MailSec. Possible sequences of dialogue moves are defined using transition networks. The system uses syntactic and semantic features of the user's utterance together with the current game state to recognize a number of possible moves and a scoring function is used to determine the most appropriate one. The system then selects its move from the possible transitions using another scoring function. However, if a transition path runs into a dead-end, the system can backtrack to re-select the path with next highest score. Dialogue control in the Autoroute system is similar to the state-based approach but based on abstract dialogue moves rather than specific dialogue states, thus dialogue strategies can be reused in other applications. Nevertheless, modelling dialogue move selection using transition networks is only suitable for simple dialogue interaction.

Theoretically, the information state approach is a more general version of the frame-based method, allows complex representations of the information state, and allows modelling of a variety of complex mental states such as obligations

and commitments. However, dialogue control using dialogue moves and update rules does not facilitate complex dialogue behaviour, thus in practice, this approach is not suitable for sophisticated dialogue interaction. There is still the problem of defining and maintaining a large set of dialogue moves, update rules and their rank order. However, the advantage of this approach over the simple frame-based method is to allow modelling simple goal-directed behaviour through the use of dialogue moves. This helps improve the modularity of the dialogue model and enables the reuse of some dialogue moves in other similar applications. Although, as a result, some domain-independent update rules can also be reused, most update rules involve updating the information state and require domain-specific information, which must be redeveloped for other domains. From the developer's point of view, even when extending a relatively small rule set, testing and debugging is difficult as it is not easy to understand how adding or changing a rule will affect the overall intended behaviour of the system.

Mirkovic and Cavedon [2005] have proposed a method for extension of dialogue models that are based on the dialogue move and information state update approach. A dialogue move scripting language has been introduced, which allows inheritance of existing (possibly domain-independent) dialogue move scripts for developing domain-specific extensions. Each script consists of conditions for mapping the user input to a dialogue move and rules to specify the required information updates. Inheritance in this approach is similar to the use of *interface* in the Java programming language, which helps improve abstraction but not much reuse.

2.3.5 Distributed Dialogue Models

An alternative method to the above centralized approaches is having a distributed dialogue model and dialogue management controlled by more than one system component. In the Jaspis architecture [Turunen and Hakulinen, 2001],

dialogue management is performed by three kinds of components: *dialogue agents* that are capable of handling the dialogue interaction in different ways; *dialogue evaluators*, each giving ranking scores for the dialogue agents on a specific feature; and a *dialogue manager* that controls the selection and performance of the dialogue agents and dialogue evaluators. For example, in Mailman [Turunen and Hakulinen, 2000], a telephone-based e-mail management application of Jaspis, there are specialized dialogue agents, each corresponding to a system's dialogue action such as listing e-mail messages or generating context-sensitive help. The specialized dialogue agents produce events, which are further processed by the domain-specific dialogue agents. These domain agents are responsible for interacting with the back-end to perform tasks such as retrieving e-mail messages from the server or activating an e-mail message for reading. An advantage of this approach is that it allows dialogue adaptation based on the use of evaluation functions. By adjusting the dialogue evaluators, it is also possible to change the system's strategies for selecting dialogue agents. However, this approach is only practical if the dialogue model is simple. Complex dialogue interaction requires a large number of dialogue agents, making it difficult to define the evaluation functions. Moreover, increasing the number of dialogue agents requires more computation of the ranking scores, thus affecting the system's performance efficiency. Reuse of the domain dialogue agents in other application domains is not possible. Considering the Mailman application, it is also difficult to reuse the specialized dialogue agents as they are not completely domain-independent.

The dialogue system developed at the MITRE Corporation is also based on a distributed dialogue model [LuperFoy *et al.*, 1998]. Dialogue control is performed by three components: the dialogue manager, two context trackers and two pragmatic adapters. The system is implemented using a "blackboard" style multi-agent architecture, the Open Agent Architecture [Martin *et al.*, 1999], in which the three dialogue components and other agents (e.g. speech recognizer, speech synthesizer, back-end interface, etc.) interact with each other by reading

and writing messages to the blackboard. The two context trackers maintain lists of discourse entities and resolve or insert references for input and output utterances. The pragmatic adapter on input is used for recognizing the back-end tasks from the logical forms of the user's utterances. Conversely, the conversion from back-end results to logical forms is handled by the other pragmatic adapter. The interaction among these components and other agents is controlled by the dialogue manager. Using a distributed dialogue model makes it difficult to specify and control the system's overall behaviour. Moreover, system robustness is difficult to be guaranteed in the case of an individual component's failure. Because dialogue management is handled by a set of agents, in order to maintain a coherent dialogue, there are aspects of the dialogue model that must be duplicated in multiple components. Furthermore, if both the user and the system initiate conversation at the same time, a synchronization problem may occur as the distributed nature of the system makes it difficult to coordinate simultaneous communication events.

2.3.6 Plan-Based Approaches

There has been other work on modelling dialogue for complex task domains, which views dialogue as a collaborative process in which the user and the system work together on problem solving tasks [Grosz, 1977; Allen *et al.*, 2002]. Plan-based approaches consider the speaker participating in a dialogue as performing acts to achieve goals e.g. domain tasks. Dialogue control in these approaches, therefore, is based on recognizing and planning communicative and non-communicative actions to fulfil these goals/tasks. Collagen [Rich *et al.*, 2001] is a Java "middleware" for building dialogue systems, which uses plans to model tasks, represented as a hierarchy of sequences of actions typically performed to achieve goals in the application domain. Collagen focuses on using plan recognition for identifying the user's requested tasks, constructing the task structure and managing the dialogue interaction. In Collagen, the task and dialogue

models are not clearly distinguished, being captured in a common structure. Collagen has been used in developing a number of applications such as air travel planning [Rich and Sidner, 1998], teaching procedural tasks [Rickel *et al.*, 2000], etc., most involving the system guiding the user step by step in recognizing a task plan, which does not require complex and flexible dialogue interaction. The system is not required to effectively respond to dynamically changing situations, such as for handling speech recognition errors, user cancelling tasks or the user going back to adjust information, etc. Collagen provides a common architecture for developing dialogue systems that require complex task models but constrained dialogue interaction. On the other hand, dialogue management in personal assistant applications require more than just planning tasks but providing a sophisticated level of dialogue interaction.

Ardissono, Boella and Lesmo [2000] have proposed a plan-based dialogue model which is much more complex than that of Collagen. Taking the view that humans communicate to achieve goals that must be recognized by the partners in order to maintain the coherence of the dialogue, the approach is based on modelling dialogue interaction as the process of plan formation and execution. Every utterance is expected to be coherent with respect to the ongoing dialogue, and that it must in some way contribute to one of the currently executing plans. The contribution can be either to the satisfaction of a recognized goal, an inferred goal or to the current execution of a plan. There are different levels of dialogue plans: a meta-level describing recipes for plan formation and execution, and an object-level for recognizing and executing task plans and performing primitive domain actions. The proposed computational model of dialogue is more complex than for typical dialogue applications such as personal assistants where the tasks are not complex but the required dialogue interaction is more sophisticated. If the task structure is complex, the combined process of plan recognition and planning in this approach might become intractable. The model has not been implemented in any existing dialogue systems but only an artificial domain for illustrate the proposed features. The domain task is to make a plan for an

university student to register for various types of exam such as oral exam, written exam, lab work, etc.

The work of Fitzgerald and Firby [1998; 2000] also considers natural language dialogue is a kind of task planning and execution, thus dialogue systems are instances of task execution systems. In this work, the authors propose the use of the Reactive Action Package (RAP) system [Firby, 1989] for developing dialogue applications. The architecture provides a Conceptual Memory, an assertional database, for maintaining the description of the various concepts mentioned during the conversation. These concepts are pre-specified so that the Conceptual Memory Parser, using pattern matching techniques, can extract attribute patterns from the input language to formulate its concept description. There is always a top-level task waiting for the input language. The concept description of the input is resolved to a RAP task, which is either an action or a query that can be handled by a plan. Task execution in RAP is situation-driven. A plan consists of tasks, each having a satisfaction test, a time window and a set of execution methods that are appropriate in different circumstances. If the execution of a task involves sub-tasks, the task is suspended and its sub-tasks added to the system's task hierarchy called the task agenda. The RAP interpreter selects from the agenda unsatisfied tasks for execution, starting from the top-level. If the task fails, it may be executed as many times as necessary (within the time window) to make it satisfied or immediately removed from the agenda. The RAP system was originally designed for reactive sensor-based robots, focusing on task re-planning in response to observable sensory data. However, management of the dialogue interaction requires handling not only short-term but long-term dialogue goals of the participants. Parallel plan execution in RAP makes it difficult to manage the dialogue control flow. The RAP system can provide an architecture for developing dialogue applications but this approach lacks a theoretical basis for modelling discourse phenomena in complex dialogue interaction.

2.3.7 Agent-Based Approaches

Multi-Agent Based Distributed Dialogue Model

The dialogue modelling approach used in the TRAINS system [Allen *et al.*, 1996] and its successor, TRIPS [Allen *et al.*, 2000; Allen *et al.*, 2001], is a kind of plan-based method. The underlying idea is using plan recognition for identifying the user's speech act and domain-specific goals. However, dialogue management is performed in a distributed manner, using a multi-agent system consisting of agents for handling different aspects such as speech recognition, speech generation, chart parsing, discourse management, reference resolution, verbal reasoning, problem solving, etc. Having complex dialogue and task models, these systems model human-system interaction as human collaborative problem solving, thus suitable for applications in complex domains such as an emergency vehicle dispatch scenario. Approaches such as in TRAINS and TRIPS would be unnecessarily complicated for personal assistant applications that involve less complex problem solving but more continuous dialogue interaction. In such a system, a centralized dialogue model is more appropriate to avoid synchronization problem. In addition, the dialogue manager should be light enough to be computationally effective, focusing mostly on maintaining a coherent and sophisticated dialogue.

Dialogue Manager as a Single Agent

Agent-based approaches can be considered a remodelling of plan-based methods, providing a declarative way to model the dialogue using more comprehensive formal frameworks than that of the plan-based method. Agent-based methods view dialogue interaction as rational action, exhibiting goal-directed behaviour. In these approaches, dialogue control is performed by a single dialogue manager agent. The dialogue model is explicitly encoded using an agent architecture and the dialogue structure automatically emerges according to the agent's rationality principles.

Sadek [1999] proposes the use of an agent framework for the development of dialogue agents. His model of rational agency is based on the notions of the agent's mental attitudes: beliefs (logical propositions that the agent consider true), uncertainty (what the agent believes is not yet true) and intention (the agent's commitment to bring about a proposition), and use a number of logical axioms in a first-order modal logic to model rational behavior. The framework has been used in the Artimis architecture of France Telecom, a system for developing dialogue agents. Artimis consists of a rational unit, a domain knowledge management component and a language understanding and generation component. The rational unit is the kernel of the agent, which performs reasoning about actions and plans using a theorem prover as the inference engine. In Artimis, not all stages of dialogue processing are done within the agent framework. Language understanding and generation are performed by components external to the rational unit (the system's dialogue manager) though still require information about the conversational context or the current dialogue state such as for reference resolution and generation. Hence there is a practical problem integrating the agent's kernel with the external components. Most dialogue applications developed using Artimis are natural language database query systems such as for accessing weather or transportation information.

Wallis *et al.* [2001] have presented an initial proposal for the use of the BDI (Belief, Desire, Intention) agent architecture of Rao and Georgeff [1995] for discourse analysis. In this approach, dialogue interaction is viewed as consisting of goal-directed dialogue games which can be handled by the agent's dialogue plans. To illustrate the approach, the car booking domain is chosen and dialogue plans for this domain are built using a task analysis method. Then 25 real dialogue phone calls were recorded, transcribed and analysed to understand the errors made by users and the strategies used by the expert. A set of plans (or procedures) was built based on the result analysis of the task structure. Each plan is associated with a goal such as taking call, entering user name, etc. There can be several plans for handling a same goal, but only one that matches the

dialogue cues will be selected for execution. The authors have claimed that this set of plans can be directly mapped into the BDI plan structure to obtain a dialogue system for booking cars, nevertheless this has not been done.

Our approach falls into this agent-based category but differs from these existing methods in several respects, making it more suitable for personal assistant applications. Dialogue modelling for domains such as e-mail and calendar management has additional requirements to allow more complex and sophisticated dialogue interaction. The system must maintain a mixed-initiative and coherent conversation with the user, and additionally, tailor the responses to the conversational context, the user's preferences, etc. Thus the use of information about the conversational context and other knowledge is required in various stages of dialogue processing. These processes should therefore be tightly integrated in a common framework (e.g. a centralized dialogue manager agent) to enable more effective management of the dialogue. The approach of Sadek does not facilitate this centralized management requirement. The work of Wallis *et al.*, on the other hand, is focused on discourse analysis, building dialogue models based on collected training examples. In our case, more robust and active dialogue management is required to provide natural and flexible interaction.

2.4 Spoken Dialogue Management

Spoken dialogue management requires additional consideration to provide natural and sophisticated interaction. Firstly, good dialogue control strategy should allow both the user and the system to take the initiative in the conversation. The user should not be forced to give an exact answer to every system prompt, but be able to provide more information than as required, or at any time cancel the current request and start a new one. Support of user barge-in may be preferred to allow immediate cancellation of the ongoing task or interrupt a long response. User barge-in, however, can cause problems for speech recognition. Another im-

portant issue is whether the system should or should not interrupt the user when the user is speaking. The system should take into account the importance of the notification and the preferences of the user. Smith and Hipp [1994, pp. 11-12] have suggested four different levels of system initiative, which can be adjusted according to the application domain and the user's competency level to provide more effective interaction:

- *Directive*: The system has full control of the dialogue interaction, prompting the user for every missing piece of information. The user is only expected to provide short and simple answers. This kind of interaction could be suitable in travel booking, route finding and similar applications.
- *Suggestive*: The system still has dialogue control, but suggests and allows the user to change the direction of the dialogue according to the user's preferences. The user can provide as much information as he/she wishes. This suggestive mode can allow more flexible and natural dialogue interaction.
- *Declarative*: The user has dialogue control. Although not required, the system is able to mention additional relevant facts, in responding to the user's questions. This mode of interaction could be used in simple question-answer type services.
- *Passive*: The user has complete control of the dialogue. The system responds passively and directly to the user's questions. This simple mode of interaction is only suitable for command-based speech-enabled interface such as for operating home appliances.

A conversational dialogue is achieved by the mutual exchange of utterances between the participants. Thus the system should always indicate that the conversation is ongoing and that the system has perceived the user's requests. If the requested task has been completed but does not have observable effects, the system should state that the task has been done. If a task takes long to accomplish, there should be an indication that the task is in progress. In addition,

the system should have a time-out so that if the user does not respond after a while, the user is re-asked the last question or is suggested to request another task. Grounding is often required in conversation to establish a common ground of understanding among the participants. In spoken dialogue applications, misinterpretation can occur due to speech recognition or dialogue processing errors, which may not be easily noticeable to the user as in human-human dialogue. Hence the system should make clear to the user how it understood the user's request, for example, by repeating important information in the response (e.g. proper names, temporal phrases, etc.) or confirming non-reversible tasks.

Generating appropriate responses also contribute to the satisfaction of users of dialogue systems. A question is whether the system should behave naturally, adopting some human-like characteristics, e.g. answering the user with complex sentences and in different ways, or whether it should maintain a consistent, direct but uninteresting style. Perhaps many users do not like the idea of talking to a machine because current telephone-based dialogue systems can only provide a very simple kind of interaction. On the other hand, varying system responses may confuse the user. The user may over-estimate the system's supported level of dialogue sophistication, resulting in system failures and user frustration and dissatisfaction. Another point for consideration is that novice and experienced users may have different expectations in using the system. It is useful if the system can use the context to detect any problems and provide appropriate help messages to guide the users. Finally, a speech synthesis component capable of producing more natural voice output can help achieve better user satisfaction.

2.4.1 Overcoming Speech Recognition Limitations

Successful development of spoken dialogue systems requires the integration of different language technologies including speech recognition, speech synthesis, language processing, dialogue modelling, etc., in which the system's performance

relies heavily on the performance of the speech recognition component. One fundamental problem with spoken dialogue systems is that the interaction can easily fail due to speech recognition errors. Although speech recognition systems have significantly improved recently, the performance is still far from perfect. Hence dialogue models must include mechanisms for recovering from these errors to provide an adequate level of system robustness. There are three type of speech recognition errors: deletion, substitution, and insertion errors. Deletion errors occur when the user says some words but they are not recognized at all by the speech recognizer. Substitution errors correspond to the case when some words are misrecognized as other words. In the case of insertion errors, the speech recognizer's output includes words that the user did not say. Substitution errors cause more problems for dialogue processing than others.

In comparison to speech recognition in dictation mode, grammar-based speech recognition can give better accuracy. However, grammar-based speech recognition components are likely to exhibit performance degradation as the vocabulary size increases. Thus their users are expected to use more structured utterances and are limited to a set of pre-defined words. Moreover, constructing grammar rules for a grammar-based speech recognition system can be difficult since often it requires the use of a large number of collected training examples. However, a grammar-based method is more effective for constrained application domains such as travel booking or route directory services, in which dialogue interaction can be made simple and direct, and proper names, i.e. location names, can be pre-specified. Complex and unconstrained dialogue applications, on the other hand, are better with speech recognition in dictation mode because although many domain-specific keywords can be pre-specified, other phrases, and so the vocabulary, should not be restricted. These speech recognition systems have a potential for covering a wider range of spoken utterances, even those that are not grammatically correct. It is also worth mentioning that user-dependent speech recognizers require a period of training, while other speaker-independent systems require no training but support only limited vocabulary.

In spoken dialogue systems using off-the-shelf speech recognition software, it is impossible to detect and correct speech recognition errors directly in the speech software. Error recovery is therefore usually done as part of dialogue management and at the level of pragmatic analysis, possibly using the constraints provided by the conversational context or information from the back-end applications. In the domain of personal data management, an example is the use of back-end information such as the user's address book to constrain the possible proper names that match the user's input. The system can also detect problems with speech recognition output if the user's current utterance does not fit in with the ongoing conversation, and can suggest or prompt the user to resolve the problem. Some speech recognition software can provide a confidence measure representing the reliability of the recognition result. The dialogue system can request the user to repeat the utterance in the case of having a low confidence value. Although it may not be user-friendly, a multimodal dialogue system may allow the user to see the output of the speech recognizer so that the user can detect and self-repair speech recognition errors by repeating or rephrasing the utterance.

2.5 Multimodal Dialogue Management

Each communication modality has its own advantages and disadvantages. Multimodal systems combine different input modes such as speech, GUI actions, hand gestures, etc., and process them in a coordinated manner with multimodal output to provide greater expressive power, naturalness, flexibility and portability [Oviatt, 1999]. These systems require an effective method for time-sensitive fusion of multimodal inputs that co-occur temporally, taking into account that the information conveyed by different input modes may contain some degree of redundancy. It might be thought that incorporating multiple error-prone recognition technologies (e.g. speech, gesture recognition, etc.) will produce compounded errors, resulting in unreliable system performance. However, according

to Oviatt, multimodal systems in fact can support more robust recognition due to the user's natural ability to optimize and use multiple input modes effectively.

The existing work of McGlashan [1996] allows speech and GUI input (but not both in the same turn) for accessing a consumer information service about microwave ovens. The dialogue modelling approach used in this work is information state based, in which the dialogue model is composed of contextual variables, dialogue goals and corresponding rules for updating these variables. There are domain-dependent heuristic rules, which are based on the type and content of the system goals, used for modality selection in responding to the user. The system also employed goal-specific strategies in deciding to offer the user more or to filter out information when the user's request results in too few or too many solutions that can not fit on the screen.

SmartKom [Wahlster, 2006] is a large cooperative project conducted by major German research institutes, companies and universities over a four year period, which has resulted in an architecture for multimodal interaction. The SmartKom architecture [Herzog and Reithinger, 2006] consists of many functional blocks, each for handling a part of the dialogue processing task and interacting with each other by exchanging messages in XML format. In SmartKom, the modality fusion component [Engel and Pflieger, 2006] is responsible for merging contents of the input modalities into a unified representation, i.e. typed feature structure or frame, which is then used to identify the user's intention. The dialogue model of SmartKom is based on the information state approach, representing the user's intention as a sequence of dialogue goals/moves. Multimodal fusion in SmartKom is done by treating speech modality as the primary input mode. Referring expressions specified in the user's utterances are then resolved by considering information of other appropriate input modes.

For personal assistant application such as the SPA, providing a combination of spoken dialogue and a graphical user interface on devices such as PDAs allows more efficient user-system interaction and enhances the system's usability. Even though such systems need to support only limited multimodal input and output,

dialogue management still requires timely handling of the speech and GUI input. Taking an example in the e-mail management domain, the user might select several message headers on the PDA's interface, then use speech to request those messages be archived to a folder. The message list must be added into the conversational context before the spoken request is processed. In addition, the system's responses may need to be tailored to the available modalities. For instance, a small calendar GUI, only capable of displaying a brief summary of the user's appointments, should be combined with more detailed speech output. On the other hand, if the GUI can display more details of the appointments, a short speech output would be more appropriate to avoid redundancy.

2.6 Concluding Remarks

We have presented in this chapter important work in speech act theory and discourse analysis, which take the view that speaking language is performing communicative acts, according to the speaker's intentions, in order to achieve goals. In addition, the coherence of a dialogue depends on the recognition of the speaker's intentions, which can be done by firstly, recognizing the performed speech acts. The linguistic concepts and computational models presented in this chapter have potential application for use in dialogue modelling in personal assistant applications. We have also reviewed in this chapter the existing work in dialogue management, discussing advantages and disadvantages of these methods. In order to provide robust and sophisticated dialogue interaction, we propose an agent-based approach, which is suitable for developing personal assistants for multi-domain applications such as the SPA. Details of our agent-based approach will be described in the following chapters.

AN AGENT-BASED APPROACH TO DIALOGUE MANAGEMENT

In this chapter, we describe our agent-based approach to the problem of dialogue management in software personal assistants such as the Smart Personal Assistant (SPA). The SPA is an integrated collection of back-end task assistants, allowing the user to remotely and collectively perform tasks in application domains such as e-mail and calendar management through a unified user interface. Moreover, with the availability of speech modality on mobile devices such as PDAs, the SPA enables user-system interaction to be conducted via a sophisticated spoken natural language dialogue.

We will explore in this chapter the dialogue requirements in the case of the SPA and give arguments for our motivation to employ an agent-based approach to dialogue modelling and control. We argue that dialogue management in the SPA should exhibit agent-like behaviour, which motivates the use of a BDI agent architecture for developing the SPA's dialogue manager agent, in which the dialogue model is explicitly represented in the agent plans and beliefs. The dialogue manager agent uses its beliefs for maintaining information about the dialogue states, including the discourse history and the salient list, as well as the domain-specific knowledge and the user model.

Next we will give an overview of BDI agents and then, describe in detail our plan-based dialogue model, which consists of different (agent) dialogue plans for handling different discourse-level and domain-level goals. Moreover, dialogue processing is done automatically as the result of the agent's BDI interpreter

selecting and executing plans during its execution cycles, according to the agent's beliefs and current context. The modularity of the plans allows the dialogue model to be extended and reused in other applications.

In the next chapter, we discuss dialogue management using our agent-based approach in the context of the SPA application. We shall present the implementation of the dialogue manager agent using the JACK Intelligent Agent platform [Howden *et al.*, 2001], a Java-based PRS-style system. In addition, the course of dialogue processing in the SPA will be described in detail through dialogue examples.

3.1 Dialogue Characteristics

To begin with, we describe again our application, the *Smart Personal Assistant* (SPA), and then, analyse the requirements of dialogue management in the SPA and similar applications. The SPA is a suite of task assistants, each an agent “wrapper” around existing task management software. Each task assistant specializes in a particular application domain such as e-mail or calendar management. The e-mail assistant supports tasks such as searching, deleting, archiving and replying to e-mails. In addition, the user is able to request for notification of important message arrivals. Similarly, the calendar assistant supports common calendar tasks such as scheduling appointments and/or other user activities. The integrated collection of the tasks assistants, i.e. the SPA, is accessible via a range of devices, e.g. PDAs, desktops. Importantly, the user should be able to interact with the system through a spoken natural language dialogue and the GUIs of the devices.

The SPA application requires both coordination of the assistants and management of the spoken dialogue with the user. Coordination of the task assistants is required to provide a single point of contact for the user to conduct a coherent dialogue with all the assistants and be able to easily switch between different front-end devices. In terms of dialogue management, the system needs

to be able to recognize the user's intention when doing actions on the interface or speaking natural language commands, which may involve acquiring domain information from the back-end assistants. In addition, the dialogue responses should be tailored according to the conversational context as well as the user's physical context, devices and preferences.

Coordination Requirements

The user should not have to deal with each assistant separately but be able to interact with them as if there is just a single smart assistant. An excerpt from a dialogue scenario is given in Figure 3.1, in which a user changes the conversational context from e-mail to the calendar domain, making references to the same person, John Lloyd, and suggesting a new appointment. After the new appointment has been successfully created, the conversation is shifted back to the e-mail domain. A single point of contact has been provided that allows the user to easily switch back and forth between different task assistants while still being in a continuing conversation with the whole SPA.

During a course of interaction, the user may change physical context by switching to different devices although it is unlikely that more than one front-end device would be used at the same time. It is required that the interaction and conversational dialogue with the user will be maintained coherently. However, the user's preferences in interacting with the SPA may not be the same in different physical contexts and on different devices. Thus the user's current context, e.g. location, devices, etc., need to be taken into account in responding to the user. In other words, the SPA should be able to tailor the system responses according to the conversational context and the user's physical context, devices and preferences.

Moreover, in some cases, complex interaction with the back-end assistants is required in fulfilling the user's requests. For example, for the task of searching e-mail messages about a certain topic, only looking for keywords in the message contents might not give the best results. In the SPA, the e-mail assistant is

User Do I have new e-mail from John?

SPA You have two new messages from John Lloyd.
One is about slides and one is about workshop.

User Show me the one about the slides please.

SPA *Displays the message from John Lloyd about slides.*

User I need to see him at 5 pm this Friday.

SPA Do you want to create that appointment?

User Yes, please.

SPA What is the appointment title?

User Workshop.

SPA Appointment has been created for Friday 5 pm.
Displays calendar view showing the new appointment.

User Any messages from Paul?

SPA You do not have any new messages from Paul.

User Notify me if I have an e-mail from him.

SPA Certainly, I will notify you if you have new messages from Paul.

Figure 3.1: Dialogue Illustrating Context Switching

capable of classifying messages into different categories, thus the system can make use of the classification results in the search task. Processing the domain tasks may even involve combining information from more than one assistant, which requires coordination of the task assistants. Furthermore, the system needs to correctly handle two-way information flow, such as when notifying the user of new message arrivals. This strengthens the point made earlier that the dialogue manager of the SPA must be the central component for maintaining a coherent dialogue with the user as well as coordinating the activities of the other assistants.

Dialogue Management Requirements

Leaving the coordination requirements aside, dialogue management involves maintaining the conversational context and having strategies for controlling the

conversation flow. The dialogue in the case of the SPA is mainly user-driven. Nevertheless, system initiative is also essential for clarifying user requests or notifying the user of important events. Hence the dialogue management, from an agent point of view, requires some degree of *pro-activeness* for error recovery but also *reactiveness* in order to fulfil the user's requests. Therefore, the SPA requires a sophisticated dialogue model for handling such complex behaviour.

Firstly, the system must understand the user's intentions when doing actions on the interface or speaking natural language utterances. Information from the back-end assistants must be exploited in interpreting the user's requests. For instance, the sample dialogue given in Figure 3.2 illustrates a possible use of back-end information in the calendar's address book to resolve the reference of "John" in the user's utterance.

User	Do I have any appointments with John tomorrow?
SPA	<i>Searching the address book of the calendar assistant has found one John, viz. John Lloyd.</i>
	You have an appointment with John Lloyd at 3 pm tomorrow about workshop slides.
	<i>Displays calendar view showing the appointment.</i>

Figure 3.2: Use of Back-End Information to Support Dialogue Management

Secondly, from the developer's point of view, the dialogue model of the SPA should be extensible; reuse and maintenance should not take much effort. A multi-domain dialogue system would likely require the addition or removal of domain-specific back-end components, and the extension or modification of the dialogue model for handling different domains. Thus the SPA's dialogue model should allow extensions to cover new tasks and/or more dialogue aspects. Moreover, the dialogue model should be scalable, enabling the integration of new task assistants or being customized to a different set of task assistants.

Although the dialogue model of the SPA must be user-independent so that the system can be deployed for different users, it should be easy to incorporate adaptation methods into the model so that the dialogue can be adaptive to the user's device, physical context and preferences. For example, the system must decide not to use spoken dialogue in environments such as seminars since it would be socially inappropriate. Similarly, the user's current location and activities could be made available from the calendar entries or from sensor data. As discussed previously, more interesting is the potential for the character of the dialogue to change during a course of interaction. The user may change physical context by moving around and/or being connected to new devices. The same back-end assistants can be used to avoid synchronization issues such as data inconsistency. However, the interaction and conversational dialogue with the user must be maintained coherently, which require that the system responses be tailored to the current device. For instance, while full information can be shown on large-screen devices (e.g. desktops), only important information should be displayed on small devices such as PDAs. Instead of showing the whole content of a message, a summary containing the main information could be given.

Motivation for an Agent-Based Approach

As we have discussed in Section 2.3, most existing spoken dialogue systems focus on simple and highly constrained tasks such as telephone-based flight and travel booking or transportation timetable services. These applications employ relatively simple dialogue models, in which the representation of dialogue states is based on a finite state machine or a hierarchy of frames or scripts. Dialogue control is governed by a set of pre-defined rules. State-based dialogue models are very limited because of their small number of states and the high cost required for dialogue extension. The major disadvantages of the frame-based and script-based methods is that maintenance of the system's frame/script hierarchy and/or rule set requires significant effort. Although some of the rules or frames/scripts can be reused, extending the dialogue models for handling additional task domains is difficult because of the lack of modularity. Moreover,

appropriate priorities among dialogue control rules are crucial but difficult to specify and maintain. These approaches are not suitable for applications such as the SPA. They typically cannot provide the desired level of sophistication required for complex domains such as remote e-mail and calendar management without becoming unmanageable. In the case of the SPA, dialogue interaction is more complex and mixed-initiative, in which several user and system turns are often required to complete a single task.

There has been other work on modelling dialogue for complex task domains, in which dialogue is considered a collaborative process where the user and the system work together for some problem solving tasks [Grosz, 1977; Allen *et al.*, 2002]. These applications require a very complex domain model (as for a planning system). The SPA application domain involves less problem solving but more continuous interaction than in those systems. Hence these approaches would be unnecessarily complicated for the SPA. The dialogue manager of the SPA should be light enough for this problem, and focus more on the user's interaction than on problem solving.

One of the SPA's dialogue management requirements is being reactive as well as pro-active in fulfilling the user's requests, hence we believe that it is advantageous to design the dialogue manager as an agent, specifically, using a BDI agent architecture (Section 3.2). The idea of employing an agent paradigm is strengthened by existing research, which considers dialogue interaction as rational action, exhibiting goal-directed behaviour [Perrault *et al.*, 1978; Allen, 1979; Cohen and Perrault, 1979; Cohen, 1981]. In our agent-based approach, the dialogue manager agent will have a number of plans each specifying the agent's actions for achieving a goal, such as utterance interpretation, identifying the user's intention, response generation, etc. The agent looks through its plans to find those that are relevant to its goal and applicable to the situation and executes them. The dialogue manager agent maintains coherent interaction with the user as well as being a coordinator that directs the actions of the specialist assistants such as e-mail and calendar agents.

3.2 BDI Agents

Although the term *agent* has been used popularly in the research community, its meaning is often vague, possibly because there has not been a widely accepted answer for the question of *what constitutes an agent*. The Oxford English Dictionary¹ provides two meanings of the term *agent* that are relevant to the answer of this question. In one sense, an agent acts in a particular direction to produce an effect, thus exhibiting goal-oriented behaviour. Another sense views an agent as one who acts for another, on behalf of another. In computer terms, this second meaning is close to the notion of a personal assistant that collaborates with and supports the user in various ways, such as hiding the complexity of difficult tasks, performing tasks on the user's behalf, helping the user to manage his/her own activities or coordinating teamwork activities with other users [Maes, 1994]. In this research, the SPA application as a whole could be considered a personal assistant in the second sense.

The term *agent* in this thesis, however, is used in the first sense, that is an entity situated in an environment, capable of exhibiting autonomous goal-oriented behaviour. Wooldridge and Jennings [1995] suggest that there is a weak notion and a stronger notion of an agent. An agent should have at least the two properties of the weak notion, i.e. reactivity and pro-activeness. The agent should perceive and reactively respond to the changes in the environment. However, it should also be pro-active, taking the initiative in achieving its goals. The stronger notion of agency considers the agents having other human characteristics such as informational attitudes to maintain information about the environment (e.g. knowledge or belief) and pro-attitudes to guide the agent's action (e.g. desires, goals or intentions). Many agent architectures in the literature follow the well-known BDI model [Bratman, 1987], in which an agent has three primary mental attitudes of Belief, Desire and Intention. These attitudes respectively represent the informational, motivational and deliberative states of

¹Oxford University Press, <http://dictionary.oed.com>, 2005.

the agent, Rao and Georgeff [1995]. The beliefs of the agent represent information about the world or the environment. Its desires represent the agent's desired end states. The chosen desires that the agent has committed to achieve are its intentions.

3.2.1 The PRS Agent Architecture

One instance of BDI agent architecture, the PRS (Procedural Reasoning System) architecture, was developed by Georgeff and Lansky [1987] and Georgeff and Ingrand [1989]. The PRS abstract architecture is presented in Figure 3.3. PRS consists of the current *beliefs* or facts about the world; a set of current *goals* to achieve; a *library of plans* describing the sequences of actions may be performed to achieve certain goals or react to situations; and an *intention structure* containing the plans that have been chosen for execution, either immediately or at some later time, and the partially executed plans.

The inputs to the agent are events from the environment and the outputs are the agent's actions. The *interpreter* loops and generates an action in each cycle. External and internal events are always added to an event queue. The agent's beliefs are adjusted according to those events. At the beginning of a cycle, plans are triggered from the plan library which specify courses of action that may be undertaken in order to achieve the agent's goals. Next, the deliberator, a component of the interpreter, selects a subset of these plans to be adopted and adds them to the intention structure. The agent then executes one action from one plan in the intention structure. However, due to new external and internal events, the agent can reactively choose to drop intended plans and/or adopt others. The intention and goal structures are modified by dropping successful goals and satisfied intentions as well as impossible goals and unrealizable intentions. In other words, the agent is able to adjust its goals and intentions accordingly in reacting to new situations. The agent's pro-activeness is evidenced by the use of explicitly represented intentions directed towards goals, which determine the agent's selection of next actions [Bratman, 1987].

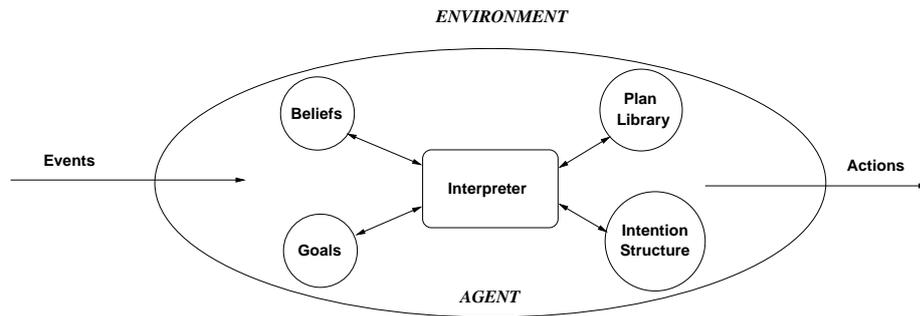


Figure 3.3: PRS Architecture

The PRS architecture has been shown to operate very effectively in continuously changing environments such as air pilot simulation [Tidhar *et al.*, 1998]. In such environments, the agent must respond appropriately in reacting to the situation as well as be deliberative in fulfilling its prior goals. The situation in the SPA application is probably less dynamic. However, the dialogue management must similarly balance the requirements of being reactive and pro-active, as has been discussed earlier. Therefore, we propose an approach to dialogue management in which the PRS agent architecture is used for developing the dialogue manager. The dialogue model is encoded in the agent’s plans which are modular units for handling dialogue and domain-specific aspects. In this approach, dialogue management strategies are also encoded in the agent’s plans so the control flow emerges automatically as the result of the agent’s plan selection by the BDI interpreter during its execution cycles.

3.3 The Dialogue Manager Agent

In this section, we present our Dialogue Manager, which has been developed using an agent-based approach. In particular, we employ a BDI agent architecture, the PRS system, for dialogue modelling and management. In our approach, the dialogue model is encoded in the plans of the Dialogue Manager agent. Each plan is a modular unit, handling a discourse-level goal such as recognizing the user’s intention or a domain-level aspect such as performing a domain task. Thus there is a separation between discourse-level and domain-level plans, which enables the

reuse of discourse-level plans in other applications. The Dialogue Manager agent maintains the conversational context and other domain-specific knowledge as its internal beliefs. Using this approach, dialogue processing is done automatically as the result of the BDI interpreter selecting and executing plans according to the current context. Before going into the Dialogue Manager’s plan-based dialogue model in detail, we discuss the use of conversational acts in our approach for the representation of the dialogue information state and handling dialogue characteristics such as turn-taking and grounding. In addition, we will briefly present the belief representation of the Dialogue Manager agent.

3.3.1 Conversational Acts

In conversational personal assistant applications, dialogue management requires not only recognizing and handling tasks that are expressed in individual user utterances but more importantly, maintaining a coherent dialogue and being able to recognize tasks that spread over multiple utterances. It is therefore necessary to understand the structure of a dialogue (or discourse) and the notion of a coherent one. As we have discussed in Section 2.1.2, there are two major lines of approaches to discourse research, which are referred to as *informational* and *intentional*. On one stream, informational approaches consider the coherence of discourse to derive from semantic relationships between the information expressed by successive utterances. We, however, follow the intentional approaches, which are consistent with early work on speech act theory by Austin and Searle (Section 2.1.1). The speaker in a dialogue, by making an utterance, intends to perform some action known as an *illocutionary act* (generally termed a speech act). According to Searle, illocutionary acts can be classified into five major classes: assertives, directives, commissives, expressives and declarations. More important is that the intention of the speaker can be recognized from the speech acts being performed. Thus, maintaining a coherent dialogue requires understanding of the user’s underlying intentions when speaking utterances by first, recognizing the associated speech acts.

Our approach is based on the theory of speech acts, so that in order to recognize the user’s intentions, the Dialogue Manager agent needs to first recognize the associated speech acts being performed. According to work in discourse by Grosz and Sidner [1986], each discourse segment has a discourse segment purpose or discourse segment intention. The *intentional structure* of the discourse contains these segment intentions and also captures the relations between them. Discourse intentions differ from other kinds of intentions that they are intended to be recognized. The speaker, in communicating an utterance, may have other intentions that are intentionally hidden from the hearer. However, when interacting with software personal assistants, there should not be any hidden intentions of the user, but only the intentions of requesting tasks, which are expected to be recognized. These tasks, hence, can be considered the dialogue’s discourse segment intentions. Moreover, with the level of complexity of the application domains of software personal assistants, we make an assumption that although there may be multiple speech acts being performed in any given utterance, only one of them is important in determining the user’s intention in the current segment. We call this act a conversational act (c.f. illocutionary act, as in speech act theory). We separate domain-independent conversational acts from domain-dependent task goals, thus although there are multiple conversational acts exchanged in each discourse segment, the intention of every segment is to fulfil a domain task. In addition, each user utterance in the dialogue plays one of two roles: specifying the user’s intention in the current segment (i.e. a task to be performed) or adjusting the recognition of that intention (e.g. clarifying or giving more information).

A description of the defined conversational acts is shown in Table 3.1. Note that in carrying out the domain tasks, the Dialogue Manager may also interact with other personal assistants. The interaction between the Dialogue Manager and personal assistants is also modelled as performing conversational acts such as a **Request** for delegating a task or a **Respond** to return the task’s results.

Conversational Act	Act Description
Greet	express the speaker's greetings and/or feelings
Request	ask the addressee to fulfil a request
Respond	describe the results of the request to the requester
Notify	notify or inform the addressee of a fact
Request-Clarify	ask the addressee to clarify some ambiguities
Clarify	clarify some ambiguities
Request-Confirm	ask the addressee to confirm by expressing agreement or disagreement
Confirm	confirm some proposition by expressing agreement or disagreement
Cancel	cancel the current discourse segment and start a new one
Advertise	special act for the personal assistant to describe its capability to the Dialogue Manager
Ack	express acknowledgement

Table 3.1: Conversational Act Descriptions

It is important to consider that dialogue (spoken and written) differs from other kinds of discourse in some characteristics such as turn-taking, grounding, etc. The proposed dialogue model allows mixed-initiative interaction, i.e. the user and the Dialogue Manager can alternately take turns in the conversation. The user takes turn when requesting tasks or confirming information while the Dialogue Manager takes the turn when requesting clarification or giving back the task results. However, the Dialogue Manager may also initiate a conversation to notify the user of important events such as e-mail message arrivals or calendar appointments that are due soon, etc. This mixed-initiative behaviour occurs automatically as the result of the Dialogue Manager agent selecting appropriate plans in a certain context.

Grounding in the literature is commonly understood as a process through which the speaker and the hearer constantly establish common ground, proposi-

tions that they both mutually believe, Clark [1996]. This is done by the hearer acknowledging the speaker's utterances or specifying any problems arising in reaching the common ground. The user needs to know what has happened if his/her request fails: has the speech recognition failed; or is the utterance too complex to be understood and needs to be rephrased; or is the request not supported by the system? Appropriate acknowledgements are always necessary to help the user determine what went wrong. Even if the request has succeeded, a confirmation of a delete task such as *"The messages have been deleted"* is desirable. In dialogue systems, grounding is required to support sophisticated and natural interaction. In our model, the **Ack**, **Request-Clarify**, **Clarify**, **Request-Confirm** and **Confirm** conversational acts are used for grounding. The **Ack** acts are for acknowledgement while the others are for resolving ambiguities. Moreover, context-sensitive system responses can also be used to provide grounding information.

Table 3.2 shows an example dialogue together with the corresponding conversational acts. In this scenario, the user requests for information about appointments with Paul on the next day. Since there are no appointments with any Paul other than Paul Compton on the next day, the system simply returns the results without the need to clarify which Paul. However, the system has mentioned the person's full name, i.e. *"Paul Compton"*, in the response for grounding. Thus any future references to either "him" or "Paul" will be resolved to Paul Compton until new entities are introduced. The user then asks the system to cancel one of the appointments. In this current context, an appointment at 2pm does not match any of the two appointments with Paul Compton, so the system performs a **Request-Clarify** act to request the user for clarification. As the user clarifies that the appointment at 3pm should be cancelled, the task can then be completed successfully and the system performs a **Respond** act to report the outcome. The two utterances in shaded rows illustrate how the **Request-Confirm** and **Confirm** acts are performed.

3.3 The Dialogue Manager Agent

Dialogue	Conversational Act
System: Hello, how can I help you.	Greet
User: Do I have any appointments with Paul tomorrow?	Request
System: You have two appointments with Paul Compton tomorrow. One at 10 am about AI workshop and the other at 3 pm about agents course.	Respond
User: Cancel the appointment at 2 pm.	Request
System: Sorry which appointment with him do you want to cancel?	Request-Clarify
User: I mean the one at 3 pm.	Clarify
System: Are you sure?	Request-Confirm
User: Yes.	Confirm
System: Appointment has been deleted.	Respond

Table 3.2: Example Dialogue and the Associated Conversational Acts

According to Grosz, Weinstein and Joshi [1995], the *attentional state* of the discourse has two components for modelling the changes of attention within the local discourse segment and at the global level, which correspond to the coherence of the discourse at the local and global level. In addition, the attentional state has an important part which contains the salient entities that have been mentioned earlier in the discourse. In our application, we refer to the attentional state as the dialogue’s conversational context, which is maintained in the agent’s beliefs. Thus the belief state of the Dialogue Manager agent contains the past and current states of the dialogue, which are represented in the *discourse history* and the *salient list*. The agent modifies its beliefs accordingly in response to the changes in the context of the dialogue, for maintaining an up-to-date conversational context. In addition, there is other *linguistic* and *domain-specific knowledge*, which is used in analysing the semantics of the user’s utterances and extracting domain-specific information for task processing. This domain-specific

knowledge is supplied by the task assistants. Moreover, there is a *user model* representing the user's current context and preferences, which can be taken into account in tailoring the dialogue responses to enhance the system's usability. In summary, the Dialogue Manager's beliefs include:

- *Discourse History*: for maintaining the conversational context such as information about the current and past dialogue states. The *discourse history* is represented as a hierarchy of discourse segments and conversational acts. Each discourse segment has a segment purpose, which is equivalent to the user's intention (or requested task) in that segment. The conversational acts are those exchanged between the user and the Dialogue Manager as well as between the Dialogue Manager and the task assistants.
- *Salient List*: for maintaining a list of objects which have been mentioned previously in the conversation, i.e. the objects that are in focus of attention. The *salient list* is used for resolving references and generating context-dependent natural language responses.
- *Domain-Specific Knowledge*: include domain-specific vocabulary and information of the tasks that are supported, which are used in interpreting the user's requests.
- *User Model*: for maintaining information about the user such as the current device, preferred modality of interaction, physical context, preferences, etc.

3.4 Plan-Based Dialogue Model

In the PRS architecture (Section 3.2.1), an agent has a set of plans, each specifying the agent's required actions for accomplishing a goal. In our agent-based approach, the Dialogue Manager is an agent with the dialogue model encoded in the agent's plans. Dialogue management requires that the user's intentions be

recognized correctly with respect to the current conversational context. If the user's intention was to request a task then the requested task must be carried out appropriately with or without interacting with the other task assistants. Finally, the outcomes of processing the task must be given to the user in suitable formats. These are the goals that the Dialogue Manager needs to achieve. We model the user-system dialogue as a set of modular plans, each associated with a dialogue aspect. Thus the Dialogue Manager controls the dialogue interaction by appropriately selecting and executing its plans according to the context. Moreover, in our dialogue model, discourse-level plans—*corresponding to domain-independent aspects*—are separated from domain-level plans—*used for performing domain tasks*. Hence, due to the modularity of this plan-based approach, there is a potential for the dialogue model, particularly the discourse-level plans, to be reused for other applications.

Figure 3.4 shows the overall structure of our plan-based dialogue model. The main part of a BDI agent are the plans that specify the actions to be performed in order to achieve its goals. In the diagram, the Dialogue Manager's plans are roughly arranged into four different groups according to their purpose: *semantic analysis*, *pragmatic analysis*, *task processing* and *response and clarification generation*. Each group itself contains several plans. In this figure, there is more than one task processing group to indicate that there is a task processing sub-group for interacting with each task assistant. We have also mentioned earlier that the Dialogue Manager uses its beliefs, i.e. the *discourse history*, the *salient list* and the user model to maintain the conversational context and the domain-specific knowledge in recognizing the user's task requests.

The inputs to the Dialogue Manager are external events arising from the user's graphical actions on the GUI or as spoken utterances become available, which trigger the agent's plan selection and execution. The execution of a dialogue plan can further bring up internal events, which in turn trigger the execution of other plans. As the system interacts with the user, the Dialogue Manager maintains the conversational context in its belief state and uses its

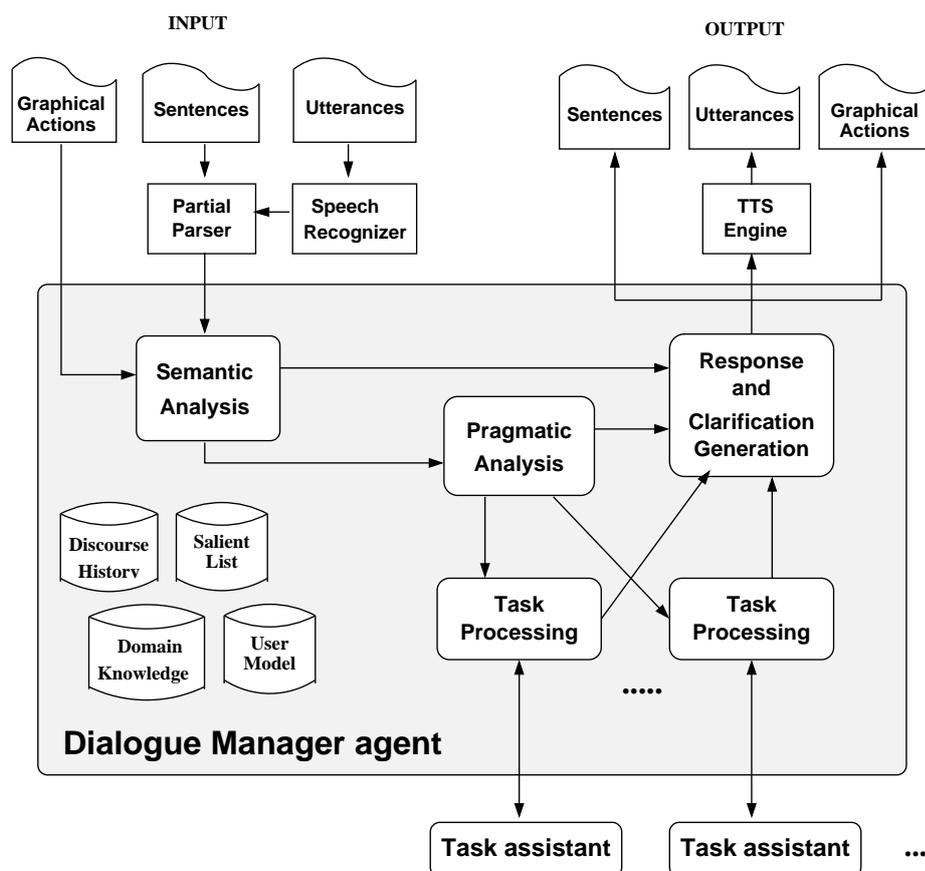


Figure 3.4: Plan-Based Dialogue Model

plans to perform semantic analysis of the user's utterances, determine the user's conversational acts, identify the user's intentions, perform the requested tasks and generate appropriate responses. An important aspect of the diagram in Figure 3.4 is that the control flow between the plans is not explicitly coded. The plans are modular units which are selected at the appropriate points by the execution of the standard BDI interpreter. The arrows on the diagram show the flow of control that arises from this behaviour so that we can understand how a series of plans are invoked in dealing with a user's utterance.

The user has the choice of interacting with the system using both or either the spoken dialogue and the system interface on the device. In some cases, it is possible that the user may prefer written to spoken dialogue in situations such as attending a seminar. Hence the Dialogue Manager is required to process not only spoken utterances but the user's actions and/or typed sentences on

the system interface. As shown in Figure 3.4, firstly the user's utterance is processed by a speech recognizer to (ideally) obtain the equivalent text. Next, this sentence is partially parsed into some syntactic representation and output to the Dialogue Manager. The partial parser uses a dictionary of keywords to parse the user's utterance or sentence into a pre-defined shallow syntactic frame. One component of the syntactic frame represents the utterance type, which can be one of the following: *greeting*, *declaration*, *imperative*, *wh-question* and *yes/no question*.

Successful management of the dialogue depends to a certain extent on the performance of the speech recognizer. In order to improve the speech recognition results, it is hypothetically possible to tightly integrate the speech recognition component with the Dialogue Manager so that the conversational context can be used to constrain the recognition problem. Nevertheless, most existing speech recognition software is commercial software, which is available "as is" and does not allow the recognition process to be constrained by contextual inputs. Moreover, it would be easier to constrain the recognition if it is grammar-based, i.e. the input utterances must comply with a pre-defined grammar in order to achieve good results. However, grammar-based speech recognition is not suitable for sophisticated dialogue applications because although domain-specific keywords are likely to be used, the vocabulary should not be restricted. Furthermore, the performance of speech recognition would decrease dramatically as the grammar becomes large. Complex domains such as e-mail and calendar management are sufficiently constrained to overcome the limitations of speech recognition over large vocabularies. However, speech recognition is required to reliably recognize keywords such as "mail", "appointment". Otherwise, recognition errors (e.g. for proper names) need to be consistent to enable correction by the Dialogue Manager, for instance with the use of back-end information such as an address book. Moreover, the dialogue model can include specific strategies to recover from speech recognition errors, such as requesting the user for confirmation or to resolve ambiguities.

The Partial Parser can be part of the Dialogue Manager but we have chosen to make it an external component of the Dialogue Manager because parsing can be done effectively without using the conversational context. Although this means any available natural language parsers could be used, complex parsing is not suitable as it will increase the dialogue processing time, hence increase the system's response time. The SPA employs a partial parser that uses scripting rules for pattern matching, using domain-specific keywords, to detect unconstrained phrases and parse the user's utterances into a shallow syntactic frame.

For each turn, the Dialogue Manager's response is generated by the *response and clarification generation* plan group and returned to the user. The response can be a task's results, a clarification question or a notification of important event. The Dialogue Manager uses the conversational context and the user's contextual information to generate multimodal responses appropriately. Speech output is generated by the Text-to-Speech Engine before being sent to the interface while text output and graphics are returned directly to the interface.

We now describe the agent plans used in our plan-based dialogue model. These plans are complex but modular units, associated with particular communicative or domain-specific goals. The modularity of this approach enables domain-independent discourse-level plans to be reused for different applications.

3.4.1 Discourse-Level Plans

The Dialogue Manager's discourse-level plans are used for analysing the user's utterances, which includes: classifying the utterances to a task domain, formulating the semantic representation of the utterances, identifying the conversational acts being performed, and recognizing the underlying intentions of the user. Each plan is a modular unit used for handling a specific domain-independent dialogue aspect. In this chapter, we intend to describe these plans at a high level of abstraction. The details of our implementation are given in the next chapter. There are roughly 20 discourse-level plans in our current implementation of the SPA.

In other applications, the number of discourse-level plans could vary, depending on the complexity of the required dialogue interaction. If the application domain requires more complex dialogue behaviours, our plan-based approach would facilitate the addition of discourse-level plans, for instance, to handle additional conversational acts. Note that the generic and domain-independent nature of these additional plans can be preserved, and so the reusability and extensibility of the dialogue model. The important point is that our approach achieves modularity at the level of plans and there is a separation between discourse-level and domain-level plans, thus enabling the reuse and extension of our plan-based dialogue model for other applications.

We list below some of the most important discourse-level plans in our dialogue model. Here, only a short description is provided for each plan as their functions can be better understood by considering the processing of dialogue, which is described in Chapter 4:

Domain Classification plan

Classify the user's utterance into a task domain by calculating the likelihood based on the occurrences of domain-specific keywords. These keywords are pre-defined in domain-specific vocabulary and kept in the Dialogue Manager's domain knowledge.

Semantic Analysis plan

Formulate a semantic representation of the user's utterance. The manipulation process is based on the general linguistic knowledge of the Dialogue Manager as well as some limited domain knowledge. This domain knowledge is supplied systematically by the task assistants so that the use of this knowledge in the plan does not affect its generic nature.

Act Type Determination plan

Determine the type of conversational act that the user is performing. This is done by considering syntactic and semantic features of the user's current utterance and the previous conversational acts performed by both the user and the Dialogue Manager. This information is maintained in the conversational context, which is kept in the *discourse history*. The diagram in Figure 3.5 illustrates the execution of the *Act Type Determination* plan. The Start, End and Fail points indicate the beginning, end and failure of plan execution. Diamonds denote decision points. Rectangles indicate calls to procedures for data manipulation. The curved rectangles denote the executions of a sub-plan.

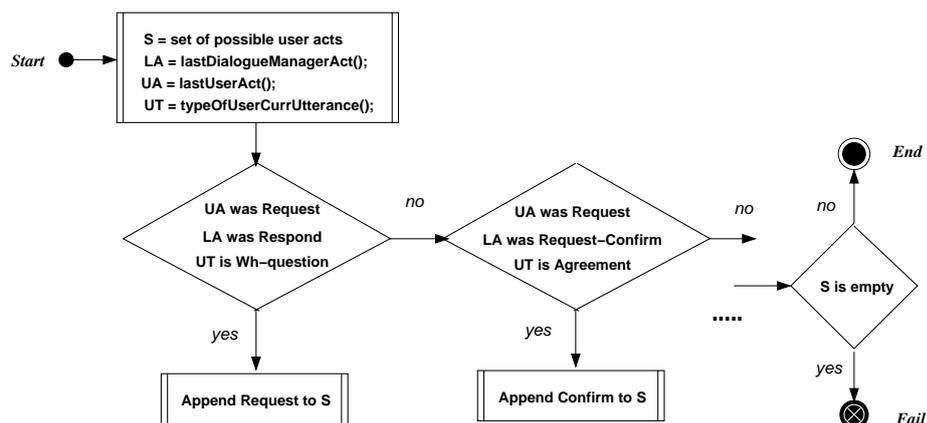


Figure 3.5: Example Act Type Determination Plan

Act Handling plan set

There is one plan for each possible type of the user's conversational acts. In executing the plan, the Dialogue Manager appropriately updates the dialogue's intentional structure (*discourse history*) and triggers execution of other plans for determining the user's intention based on the recognized conversational act. For example, if the conversational act is a *Request*, then a new dialogue subsegment will be created and the *Intention Identification* plan will be executed to determine the user's requested task.

Intention Identification plan

Identify the user's intention, i.e. the requested task. The diagram in Figure 3.6 illustrates the execution of this plan. First, a set S of the possible user's conversational acts is determined by the *Act Type Determination* plan. The Dialogue Manager selects the most likely element in this set to be the user's current conversational act CA . Depending on the type of CA , either a request must be identified or the partially recognized request in the *discourse history* must be updated. An appropriate event is raised which triggers execution of another plan for handling this conversational act. This involves identifying the type of the requested task or the attributes of the domain objects mentioned in the request. If the intention identification process fails, the Dialogue Manager selects the next element in S , and repeats this process.

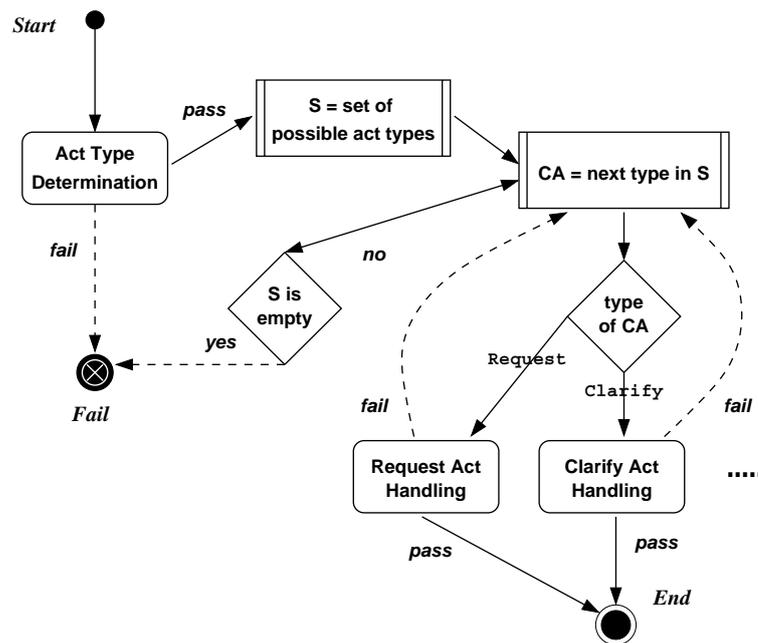


Figure 3.6: Example Intention Identification Plan

Task Type Determination plan

Using keywords in the domain-specific vocabulary to recognize the type of the requested task. Because the domain of the task has been determined, the selection

of the corresponding domain knowledge can be done automatically, allowing the plan to be domain-independent. The user's requests can also be general domain-independent tasks such as a greeting expression or a request to go back to the previous GUI screen, etc.

Graphical Action Handling plan

Process the user's actions on the GUI, which may involve performing domain tasks such as deleting e-mail messages, going back to show the previous display or updating the *salient list*.

Reference Resolution plan

Resolve anaphoric references and temporal adverbs. The resolution of definite noun phrases (of the domain objects) requires the Dialogue Manager to interact with the back-end task assistants. For example, if there are two e-mail messages in the focus of attention and the user requests *the message from John*, the two message IDs and the *from John* condition will be sent to the E-Mail agent for resolution.

People Determination plan

Determine if any contact names or references to some persons are mentioned in the user's utterance. If any task assistant provides an address book then from within this plan, the Dialogue Manager may interact with the assistant to identify a set of people that best match the mentioned name. This information can also be used to generate clarification requests. Considering an example, if the user mentions *John* where there are two Johns in the address book, the system could ask the user *Do you mean John Lloyd or John McGuire?*

Clarification Generation plan

Generate clarification or confirmation requests so that they can be directed to the user for resolving ambiguities. These questions are generated using pre-defined templates provided in advance by the back-end task assistants and kept as domain-specific knowledge.

Response Generation meta-plan

Select appropriate domain-level plans for generating system responses which are tailored to the user's device, context and preferences.

3.4.2 Domain-Level Plans

Domain-level plans are those that require domain-specific knowledge to be encoded in the plan, and include the following:

Domain Object Determination plan set

Determine conditions and attribute values of any domain objects mentioned in the user's request. This may involve combining information in the user's current utterance (such as in the case of clarification or confirmation) and in the partially recognized request. For the e-mail management domain, there is an *E-mail Determination* and a *Folder Determination* plan. An illustration of the *E-mail Determination* plan is shown in Figure 3.7.

The goal of the *E-mail Determination* plan is to determine e-mail related attributes/conditions mentioned in the user's utterance such as message is sent from John, message is about upcoming meeting, etc. Firstly, the Dialogue Manager computes the partially recognized user's intention (I), the Dialogue Manager's last conversational act (LA) and the user's current act (CA). If the Dialogue Manager's last act was a **Request-Clarify** and the user's current act is a **Clarify** then the e-mail attributes mentioned in the current utterance are

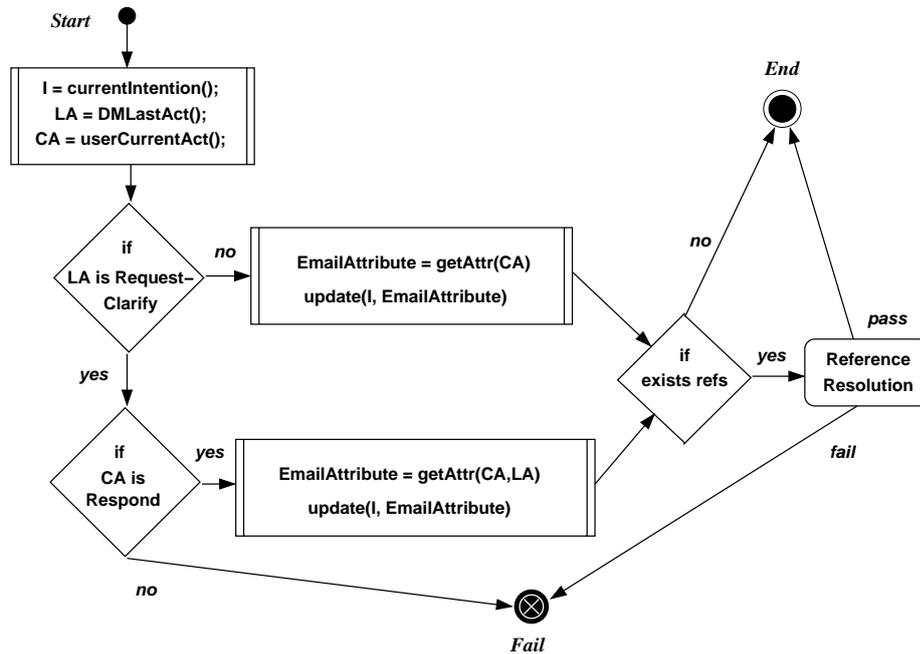


Figure 3.7: Example E-mail Determination Plan

extracted and merged with the information obtained in *I*. If there are any references to e-mail messages in the user's utterance, the *Reference Resolution* plan will be triggered for execution. If the *E-mail Determination* plan fails, it triggers the execution of the *Clarification Generation* plan, requesting the user for clarification.

Domain Task Processing plan set

There is one plan for processing tasks in each domain, which may require interacting with the back-end assistant agent such as to search for new messages, archive messages to a folder. If interaction with some back-end task assistant is required, the Dialogue Manager delegates the task to the task agent. Otherwise, other plans are triggered for immediate response generation. Figure 3.8 shows an example of the *E-mail Task Processing* plan. The diagram shows that if response cannot be generated immediately, a task is computed for being delegated to the E-mail agent. The Dialogue Manager then starts waiting for the results.

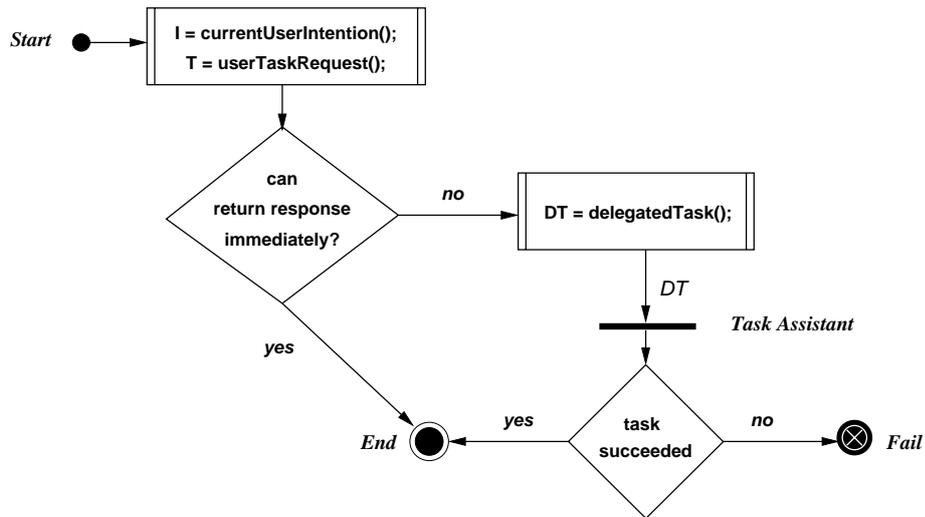


Figure 3.8: Example E-mail Task Processing Plan

Domain Task Response Handling plan set

There is one plan for processing responses from each back-end task agent. The responses may be notifications of new message arrivals or responses to the delegated tasks.

Domain Task Response Generation plan set

There is one plan set for generating system responses in each task domain so that the system responses can be adapted according to the user's device and available modalities.

3.4.3 Dialogue Model Reuse and Extensibility

Many existing dialogue systems make use of a task model and a dialogue model, however these are not always clearly distinguished. In the SPA, there is a clear separation between the task model and the dialogue model due to the use of the special Dialogue Manager agent. The SPA requires sophisticated dialogue interaction, and so requires a complex dialogue model, but not necessarily a complex task model. For our application, it is more important that the dialogue

model can be reused, so that the system can be extended without much cost, at the same time maintaining the level of dialogue sophistication.

The modularity of the SPA's dialogue model is at the level of agent plans, enabling the reuse of discourse-level plans, so that adding a new task domain requires only the development of some domain-dependent plans and domain-specific knowledge. Moreover, domain-specific knowledge is defined in a uniform format for each back-end task assistant and provided to the dialogue manager when the task assistant is connected to the system. This reduces the amount of effort required to adapt or extend the dialogue model for use in different task domains. In our initial implementation with only the e-mail task assistant, the dialogue model has about 20 discourse-level plans and 6 domain-level plans, in addition to some auxiliary plans for handling system authentication, etc. In extending the dialogue model for the calendar domain, we have reused the discourse-level plans, hence needed to develop only another 6 domain-level plans and the calendar domain knowledge description.

Although extending the dialogue model for handling the additional calendar task domain means that a new set of domain-level plans must be defined for the calendar task assistant, these plans are very similar in structure to those of the e-mail management domain. Hence defining domain-level plans for new applications does not require much effort. Even though the total number of the dialogue plans must increase as the number of integrated task assistants increases, the dialogue model is scalable and computational efficiency is not affected because most of the time the dialogue interaction is in a single domain, thus involves only the plans in that domain. For handling the calendar management domain, we have extended our initial dialogue model to include new domain-level plans, including an *Appointment Determination* plan, a *Todo Determination* plan, a *Calendar Task Processing* plan, a *Calendar Task Response Handling* plan and a *Calendar Task Response Generation* plan set. The extended dialogue model allows the user to have continuous dialogue interaction with both e-mail and calendar task assistants as illustrated in a previous example in Figure 3.1.

3.5 Discussion

Our agent-based approach is suitable for developing personal assistants for multi-domain applications. The dialogue manager agent, based on a BDI architecture, acts as the central point maintaining continuous user-system interaction and coordinating the activities of the assistants. We consider dialogue interaction as rational action, exhibiting goal-directed behaviour, thus a BDI architecture offers a complex but modular approach to dialogue modelling. The dialogue model is at high-level of abstraction, consisting of a set of modular plans, each associated with a communicative goal. Hence this facilitates the reuse and extension of the dialogue model in other applications. The dialogue manager controls the dialogue interaction by appropriately selecting and executing its plans according to the current context, which is done automatically as the result of the BDI interpreter's execution cycle.

Our approach is also beneficial in other respects. The issue of semantic interoperability is solvable by formulating declarative representations of the capabilities of individual agents and of the system as a whole, which will enable back-end integration with existing personal assistants. As the dialogue model is modular at the level of agent plans, adapting a plan-based dialogue model for integration of more specialist agents becomes easier with this approach, requiring the addition to the agent of domain-related plans and internal knowledge. Moreover, it is possible for learning to be integrated into the agent by incorporating meta-plans as intermediate steps for plan selection, thus enabling dialogue adaptation. The conversational context as well as other domain knowledge and the user model, encoded as a part of the agent's beliefs, allow the agent to easily maintain information about dialogue states and use this knowledge in interpreting the user's requests.

DIALOGUE MANAGEMENT IN THE SMART PERSONAL ASSISTANT

In Chapter 3, we presented our agent-based approach to dialogue modelling in multi-domain personal assistants such as the Smart Personal Assistant (SPA) application. We have described the use of a BDI agent architecture for the development of the dialogue manager, the central component that maintains coherent dialogue with the user as well as coordinates the back-end task assistants. The Dialogue Manager has a set of modular plans, each for handling a particular discourse-level or domain-level goal. Moreover, the conversational context and other knowledge are maintained in the agent's internal beliefs, which include the *discourse history*, the *salient list*, the domain-specific knowledge and the user model. In this chapter, we discuss the application of our approach to the particular case of dialogue management in the SPA, focusing on the dynamic aspect of dialogue processing.

We will start by presenting the implementation of the SPA application. The SPA and the Dialogue Manager agent have been developed using the JACK agent platform. JACK is a Java implementation derived from the PRS architecture, which includes mechanisms for agent communication by exchanging messages, thus facilitating the implementation of the SPA as a multi-agent system that supports interaction among the Dialogue Manager agent and other back-end task agents. Currently, the SPA has two task assistants, the E-Mail and Calendar agents, which are developed as JACK agent wrappers, built around the e-mail and calendar management software.

The BDI agent architecture in JACK provides a general framework for implementing the Dialogue Manager agent. In addition, this framework has been extended for including the dialogue data structure to maintain the conversational context. Before looking into the process of dialogue control in the SPA, we describe the implementation of the Dialogue Manager agent's belief structures: firstly, the use of domain-specific knowledge and the user model in interpreting the user's requests and generating adaptive system responses (Section 4.2.2 and 4.2.3), and secondly, the representation of discourse information in the SPA's dialogue model, in particular, the computational model of the *discourse history* and the *salient list* (Section 4.2.4).

Finally, in Section 4.3, we discuss the course of dialogue processing in the SPA, which can roughly be divided into four sub-processes: analysing the semantics of the user's utterances, identifying the user's intention, processing the requested tasks and generating system responses. In brief, dialogue control is performed automatically as the Dialogue Manager agent selecting and executing appropriate plans using the standard BDI plan selection mechanism. An example dialogue scenario will be used throughout this chapter to illustrate how dialogue control is performed.

4.1 The Smart Personal Assistant Architecture

4.1.1 Overview

Currently, the SPA has two task assistants, which allows the user to perform tasks in the e-mail and calendar management domains. The system components are as illustrated in Figure 1.2, repeated as Figure 4.1, which include a PDA interface, a speech engine, a partial parser, the Dialogue Manager agent, the E-mail agent and the Calendar agent¹. All three agents are implemented using

¹The PDA e-mail interface, the User Interface Agent and the E-mail server were implemented by Van Ho. The PDA calendar interface, Speech Processor and Calendar server were implemented by Alfred Krzywicki.

JACK and interact with each other using the JACK communication mechanism. However, only the Dialogue Manager agent makes significant use of the JACK agent's reasoning capability.

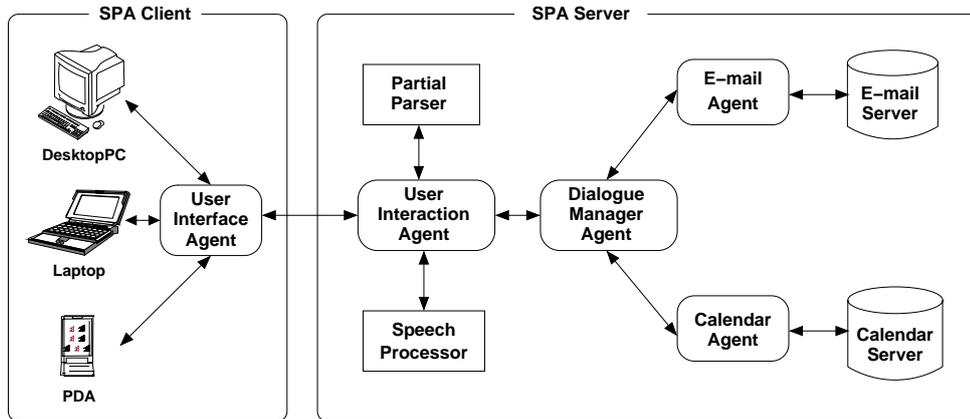


Figure 4.1: The SPA System Architecture

The user's interaction with the system is done through a simple kind of multimodal dialogue, i.e. using a speech interface as well as a limited GUI on the PDA which enables the users to navigate using a menu system or highlight important text. Thus the user's inputs to the system can be spoken utterances, typed text sentences or graphical actions on the GUI. Similarly, the outputs can be speech or GUI output. The user's spoken utterances are first processed by the speech recognition component of the speech engine, which then passes the corresponding text outputs to the Partial Parser. The typed text sentence or the word-string output of the speech recognizer is parsed into a shallow syntactic structure before being sent to the Dialogue Manager (Section 4.2.1). User graphical actions on the interface, however, are sent directly to the Dialogue Manager to update the conversational context. User requests that are initiated through spoken dialogue are handled by the Dialogue Manager while those initiated from interacting with the GUI are handled by the interface and usually require direct interaction with the E-mail or Calendar server.

In handling the user's requests, the Dialogue Manager uses information from its internal beliefs (e.g. conversational context) or may need to interact with

the E-mail and/or Calendar agent to request for information or delegate tasks. If there is no ambiguity and the request is completed successfully, the Dialogue Manager generates the responses to return to the user; speech outputs are passed to the speech synthesizer first while graphical outputs are directed to the interface. Otherwise, the Dialogue Manager generates questions to ask the user for clarification. Therefore the Dialogue Manager not only maintains a coherent dialogue with the user but also acts as a centralized coordinator agent for coordinating the task assistant agents.

The Calendar agent is implemented as a JACK agent wrapped around the calendar management software, which can be used for scheduling personal activities. The calendar management software is capable of handling common scheduling tasks such as creating, updating and deleting appointments and/or to-do tasks. The schedule data are stored in the Calendar server. All data changes made by the Calendar agent are also forwarded to the interface for synchronization.

The E-mail agent is also an agent wrapper built around the existing EMMA e-mail management software enabling the Dialogue Manager to delegate requests to the EMMA system for performing tasks such as searching, deleting and archiving e-mails, and notifying the user on the arrival of some important messages [Ho *et al.*, 2003]. In addition, EMMA can perform classification of e-mail messages in the Inbox into sorting folders, and prioritization based on the degree of importance, urgency, etc. This information is given to the user and can also be used in processing e-mail tasks. For example, the user may ask “*Do I have any new messages about meetings?*” and there exists a *meeting* virtual folder. Here the classification results of this virtual folder can be used, which is more appropriate than searching for e-mails containing the word “*meeting*”. However, the meaning of “*about meetings*” varies between users but should be consistent with their pre-defined classification rules.

4.1.2 Speech Recognition

The speech engine handles speech recognition and text-to-speech synthesis. The SPA application does not require speech recognition of 100% accuracy. However, certain domain-specific keywords (e.g. “mail”, “from”) need to be reliably recognized. Otherwise, recognition errors (e.g. for proper names) need to be consistent to enable correction by the Dialogue Manager. The first attempt used a Linux version of the IBM ViaVoice speech recognition software, which showed that although the SPA is able to recover from some speech recognition errors, ViaVoice is unpredictably unreliable. On some occasions, the results were of high accuracy while on most others the accuracy was very low. Hence, this version of the ViaVoice software is not adequate for the speech recognition task.

The current implementation of the SPA employs Dragon NaturallySpeaking in dictation mode for speech recognition and Lernout and Hauspie TTS for speech synthesis. The speech recognition is more consistent and errors more predictable, enabling correction by the Dialogue Manager. For every recognition result, Dragon NaturallySpeaking can return a number of alternative candidate phrases and their confidence values. However, our initial testing showed that most of the time, the top candidate is the best and with a very high confidence value. In addition, the implementation of the Speech Server uses the CloudGarden’s Java Speech API, which cannot return the alternative phrases. Thus we use only the top suggestion from Dragon NaturallySpeaking. Table 4.1 shows some examples of the actual utterances and the corresponding outputs recognized by this speech software using the in-built microphone on the PDA (iPaq hx4700) with no apparent benefit to using the headset microphone. As ViaVoice, Dragon NaturallySpeaking requires the user to spend time training the voice model. However, the design of the SPA system architecture is independent of the particular speech recognition engine used, thus it is possible to replace it by other user-independent speech recognition systems.

Actual Utterance	Speech Recognizer's Output
Is there new e-mail from John	Is the only new e-mails from John
Is there new e-mail from John	Is very new e-mails from John
Is there new e-mail from John	The venue e-mail from John
Show me all my e-mails from John	Show me all night e-mails from John
Show e-mail about seminars	Samuel e-mail about seminars
Move them to the seminars folder	And move them to the seminars older
Do I have an appointment with Jessica today	Do I have been appointment with Jessica today
Show it to me	Shown it to be
Show me all appointments with Jessica	Surely in all appointments with Jessica

Table 4.1: Example Speech Recognition Output

4.2 Dialogue Modelling

The dialogue management process in the SPA involves analysing the syntax and semantics of the user's input, recognizing the user's intention, processing the requested tasks and generating and returning the system output to the user's device. However, this whole course of actions is done automatically as the result of the agent's plan selection mechanism using the plans described in Section 3.4.1 and Section 3.4.2. The example scenario in Figure 4.2 will be used throughout the following sections to describe different aspects of the dialogue management process.

The Dialogue Manager has been implemented using the JACK agent platform, a Java-based PRS-like agent development platform. In the PRS architecture, the BDI interpreter loops and generates an action in each cycle. In responding to external and internal events, plans are chosen from the plan library which specify courses of action that may be undertaken in order to achieve the agent's goals. Next, the interpreter selects a subset of these plans to be adopted according to the current context. At the end of each cycle, the agent selects one action from one plan in the intention structure to be executed. Due to new

-
- 1 User Is there any e-mail from Paul Compton?
- 2 SPA You have two messages from Paul Compton. One is about project summary and one is about project proposal.
Displays the two message headers.
- 3 User Show me the one about proposal please.
- 4 SPA Here is your message about proposal.
Displays the message from Paul Compton about project proposal.
- 5 User I need to make an appointment with him.
- 6 SPA Please specify date and time of the appointment?
- 7 User Monday at 11 am.
- 8 SPA What is the appointment title?
- 9 User Project archive.
- 10 SPA Appointment has been created for next Monday 11 am.
Displays calendar view showing the new appointment.
- 11 User Show me all e-mail about seminars?
- 12 SPA You have three messages about seminars. One from Bruno Gaeta and two from Bernhard Hengst.
Displays the three message headers.
- 13 User Move these messages to the meetings folder.
- 14 SPA Meetings folder does not exist.
- 15 User Move them to the seminars folder.
- 16 SPA The messages have been moved to seminars folder.
Goes back to display calendar view.
- 17 User Do I have an appointment with Jessica for today?
- 18 SPA You have an appointment with Jessica about budget today at 4 pm.
- 19 User Delete it.
- 20 SPA Are you sure you want to delete that appointment?
- 21 User Yes.
- 22 SPA Appointment has been deleted.
Displays calendar view with the appointment deleted.
-

Figure 4.2: Example Scenario Illustrating Dialogue Management Process

external and internal events, the agent can reactively choose to drop intended plans or revise its internal beliefs. Therefore, in our approach, dialogue processing is done automatically as the Dialogue Manager agent selecting appropriate plans according to the current context, resulting from the BDI interpretation cycle. Note also that the agent's reactivity is evidenced by immediately responding to events, while its pro-activeness, by its commitment to existing goals, i.e. intentions, which constrain the agent's selection of next actions.

Figure 4.3 shows a simplified diagram illustrating the data flow between the Dialogue Manager agent and other components of the SPA. Input to the Dialogue Manager can be either the syntactic frame of the user's current utterance (obtained from the Partial Parser, Section 4.2.1) or the description of the user's graphical action on the interface. The action description includes the domain of the action (e.g. e-mail or calendar), the user's action such as `delete`, `highlight` or `archive`, etc., and the list of domain objects that are currently in focus on the GUI. For example, if the user's action was `delete` in the e-mail management domain, then the object list contains the IDs of the messages to be deleted.

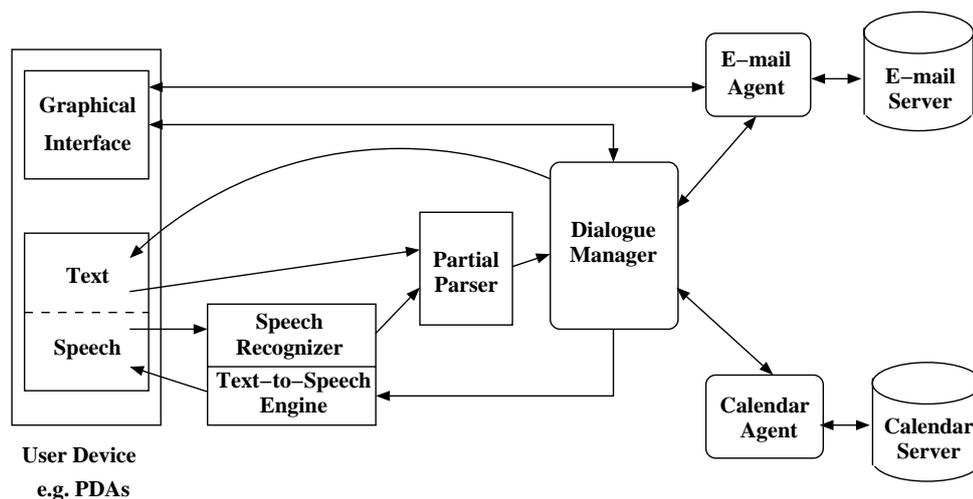


Figure 4.3: Interaction of SPA Components

Dialogue processing in our application involves, for each of the user's utterances, the Dialogue Manager recognizing the user's request, performing the

requested task (which may require interaction with other back-end task assistants) and finally, generating the response and returning it to the user's device. The user, by interacting with the interface or speaking an utterance, generates input to the Dialogue Manager, causing an external event to trigger the selection of dialogue plans for execution. The execution of these agent plans can raise internal events, which in turn, trigger the selection and execution of other plans. The behaviour of the agent on plan failure is specified in the plan, so that the Dialogue Manager agent either drops that plan or selects another one. The control flow of our agent-based dialogue management is not directly specified but derived automatically using the plan selection mechanism as part of the BDI's agent interpretation cycle. During the agent's plan execution, the Dialogue Manager agent may also revise its internal beliefs, which is how it maintains the conversational context.

For the remainder of this section, we describe the first stage in dialogue processing, analysing the syntactic structure of the user's utterances, which is done by the Partial Parser and does not involve executing a dialogue plan. We then discuss the discourse representation used in the SPA, in particular, the domain-specific knowledge, the user model, and the computational model of the *discourse history* and the *salient list*.

In Section 4.3, we shall explain how a typical course of dialogue processing is done by the Dialogue Manager selecting and executing its dialogue plans. Different sub-processes will be looked at in detail: analysing the semantics of the utterance, identifying the user's intention, processing the requested task and generating system responses.

4.2.1 Syntactic Analysis

The first stage in processing the dialogue is to parse the user utterances or typed text sentences into some shallow syntactic structure. For application domains such as the SPA, full parsing is inappropriate for the following reasons. Firstly,

existing speech software can only provide a level of quality which is far less than perfect, hence the recognition results are not likely to be grammatically correct sentences. In addition, many English speakers regularly use shortened forms of expression (abbreviated commands), which usually do not strictly adhere to the grammar. Thus, it would be inefficient or even impossible to fully parse the user's utterances. Another reason for not using full parsing is that the language vocabulary (e.g. proper names, objects) in the application is unconstrained. Hence building the dictionary for a full parser would be difficult. For example, consider the request *"Are there any new messages about ...?"*. The missing phrase could be anything. However, because the task domains are known, a domain-specific dictionary can be constructed with a reasonable amount of effort, hence the domain-specific keywords can be used for pattern matching to detect other unconstrained phrases.

The Partial Parser is implemented using ProBot [Sammut, 2001], a rule-based system embedded in a Prolog interpreter. A dictionary of keywords has been used to parse user utterances and sentences into a pre-defined shallow syntactic frame which consists of one or more clauses, each having up to eight components, as shown in Table 4.2. The keywords are classified into different syntactic categories, which are used to define ProBot rules for pattern matching. Taking the example scenario in Figure 4.2, the syntactic frame representing the user's utterance (1) is shown in Table 4.3.

Figure 4.4 shows some simple scripting rules of the Partial Parser. For example, rule (1) checks if a noun phrase is at the beginning of the utterance. If the noun phrase is found, it is assigned to the subject component of the syntactic frame and the remainder of the utterance is passed on for determining the other components. If matched, rule (2a) is for extracting an e-mail related descriptive phrase (as direct object component) and complementary phrase from the utterance. Rule (2b) extracts an e-mail related phrase and a folder related phrase as the direct object and indirect object component respectively. More example rules are given in Appendix A.

Utterance	
Component	Description
$c a e[]$	a set of syntactic frames each for a clause of the utterance
Clause	
Component	Description
$c ec e$	conjoin word used to connect clauses
e	utterance type as one of: yes/no-question, wh-question, declaration, imperative, greeting
$e e e$	an optional phrase which gives supplementary information e.g. “how many”, “when” if the utterance type is wh-question
$b ec$	the syntactic subject of the sentence
$ed ca e$	the main verb phrase of the utterance, in which the primary verb is obtained from a domain-specific dictionary, e.g. delete, reply, read, archive, create, show, view, forward, etc.
$d ec b ec$	main object of the predicate which can be e-mail or calendar related such as e-mail, folder, appointment or noun phrases
$d ec b ec$	possible second object which can also be e-mail or calendar related such as e-mail, folder, appointment or noun phrases
$c e e a e$	information e.g. time or location

Table 4.2: Components of the Shallow Syntactic Frame

User’s Utterance: Is there any e-mail from Paul Compton	
e	yes/no-question
$b ec$	there
$ed ca e$	is
$d ec b ec$	any e-mail
$d ec b ec$	from Paul Compton

Table 4.3: Example Syntactic Frame

4.2.2 Domain-Specific Knowledge

In order to provide sophisticated dialogue interaction, the language vocabulary (e.g. proper names, objects) used in the application should be unconstrained. However, it is also impractical to build and inefficient to use a large dictionary

```

(1) subject ::
    <noun-phrase> * ==> [ -- a subject noun phrase
        #assert(subject(^1))
        #goto(predicate,^2)
    ]
(2) direct_object ::
    (a) <mail-preposition> * <time-phrase> ==> [
        -- an e-mail related phrase then a complement phrase
        #assert(direct_object([ ^1, ^2 ]))
        #assert(complement, ^3)
        #goto(print_result)
    ]
    (b) <mail-preposition> * <folder-preposition> * ==> [
        #assert(direct_object([ ^1, ^2 ]))
        #assert(indirect_object([ ^3, ^4 ]))
        #goto(print_result)
    ]

```

Figure 4.4: Example ProBot Scripting Rules

for language understanding. Nevertheless, because the task domains are known (i.e. e-mail and calendar management), domain-specific dictionaries can be constructed with reasonable effort. These dictionaries provide domain-specific keywords which can be used as the boundary patterns for extracting unconstrained attribute values in the user’s utterances. For example, considering a possible phrase describing a search “*Find all new messages about school meeting*”, the system does not need to understand the logical meaning of “*school meeting*” because the presence of other domain-specific keywords such as *find*, *messages* and *about* is sufficient to recognize the user’s search request. In this search, the phrase “*school meeting*” represents the topic of interest, which can be used for information retrieval without the system understanding the meaning of the phrase.

The domain-specific knowledge also includes other information about the types of tasks that are supported by each task assistant. Moreover, for each task type, there is information about the required parameters or the kinds of objects to be manipulated. Hence in processing the user’s utterance, the Dialogue Manager can use the domain-specific vocabularies together with a description of

the object’s attributes to identify if any objects are mentioned in the utterance. Other important information is also specified in the task description, such as whether or not the task requires a confirmation. For instance, an appointment cancellation task should be confirmed by the user to avoid mistakes.

To achieve interoperability, the domain knowledge is pre-defined for each task assistant using a common format, then made available to the Dialogue Manager when required. This enables the addition and removal of task assistants to and from the SPA. In addition, this domain-specific knowledge is used to formulate task descriptions in an appropriate format for delegation to the corresponding task assistant. Figure 4.4 presents the data structure for defining the domain knowledge. This domain information could be defined more formally using ontologies, as proposed by Paraiso and Barthès [2005].

<i>F e d</i>	<i>F e d T e</i>	<i>P r o b e V a l u e - C o n f e</i>
domain	String	EMAIL, CALENDAR, GENERAL
defaultTask	String	default task type e.g. SEARCH for e-mail domain
tasks	TaskRequest[]	a set of task descriptions, each describing a domain task including the required task objects and the default question to be asked for requesting confirmation from the user
keywords	Keyword[]	a set of domain-specific keyword description

(a) DomainKnowledgeDescription Data Type

<i>F e d</i>	<i>F e d T e</i>	<i>P r o b e V a l u e - C o n f e</i>
type	String	type of the keyword e.g. NOUN, VERB, etc.
confidenceValue	Integer	a confidence value between 0 and 100 indicating how likely the utterance belongs to a particular domain given the occurrence of this keyword
synonym	String	the synonym of this word in this domain

(b) Keyword Data Type

Table 4.4: Data Structure for Representing Domain Knowledge Description

Figure 4.5 shows a part of the domain knowledge description for the e-mail management domain, which is represented in XML format, showing the mapping from a verb, “move”, to a domain task, ARCHIVE. Considering the user’s utterance (13) in Figure 4.2, “*Move them to the seminars folder*”, the main verb of the utterance’s predicate component is “move”. Thus the user’s request can be mapped to an ARCHIVE task. Moreover, from the task description, both object types, i.e. mail and folder, must be identified. The confidence value of 50 indicates that if this word appears in the user’s utterance, there is a 50% *a priori* chance (i.e. ignoring context) that the request belongs to the e-mail management domain. The reason for this value is that the same verb can also be used in the calendar domain to request a task of rescheduling an appointment.

```

<object type="DomainKnowledgeDescription">
  <field name="domain">EMAIL</field>
  <object type="TaskRequest">
    <field name="type">ARCHIVE</field>
    <field name="objects">
      <list>
        <object type="TaskObject">
          <field name="type">MAIL</field>
        </object>
        <object type="TaskObject">
          <field name="type">FOLDER</field>
        </object>
      </list>
    </field>
  </object>
  <object type="Verb">
    <field name="confidenceValue">50</field>
    <field name="word">move</field>
    <field name="synonym">archive</field>
    <field name="mappedTask">ARCHIVE</field>
  </object>
  ...

```

Figure 4.5: Example Verb-Task Mapping in Task Description

In our approach, semantic interpretation of the user’s input is performed solely by the Dialogue Manager, with the use of the domain knowledge supplied

by the task assistants. This could be done differently by delegating the work to the task agents. The Dialogue Manager could first use some minimal domain knowledge to determine the task domain, then request the appropriate task agent to perform a complete analysis of the input [Paraiso and Barthès, 2005]. However, in the case of the SPA, semantic interpretation requires access to the conversational context (which is maintained in the Dialogue Manager), thus more appropriately is handled by the Dialogue Manager.

4.2.3 User Model

The Dialogue Manager agent's belief set contains a user model for maintaining the user's current context. Different kinds of information can be modelled, including the user's current device, location, available modalities, etc. The Dialogue Manager can take into account this contextual information in responding to the user's requests. For example, if the user does not use speech, it is likely that currently, the user does not want to use speech. The reason could be that the user is in a meeting, hence the system's responses should not include speech output but only text and/or graphics. The user's preferences could also be included as a part of the user model so that more sophisticated dialogue adaptation can be achieved by tailoring the dialogue interaction according to both the user's context and preferences. The Dialogue Manager could possibly also learn the user's preferences by observing the user's interaction or explicitly asking the user through the dialogue.

Currently, the user model contains information about the user's current device and modality. In addition, the Dialogue Manager also learns and maintains the user's preferences for dialogue output in the e-mail domain, thus is able to generate adaptive dialogue responses according to the user's devices and preferences. In our approach, the Dialogue Manager agent can have different plans for handling the response generation goal, thus this learning problem can be considered the problem of learning the agent's plan selection function. We

have incorporated a learning mechanism into the Dialogue Manager agent which learns plan selection strategies. If there are a number of possible plans for generating the system responses, the user is asked to choose one among them. The chosen plan is selected and executed while the user's preferences are obtained and input to the learner for updating the learning model. Hence, gradually over time, the system will be able to learn the user's preferences and generate the system output in different formats, layouts and modalities according to the user's context and preferences. Details of our work on learning adaptive dialogue will be presented in Chapter 5.

4.2.4 Discourse Representation

Our approach to dialogue management is based on the theory of speech acts. We consider that a dialogue participant when speaking an utterance or doing a GUI action is performing a speech act. Moreover, the user's intentions (e.g. requesting a task) can be identified by first recognizing the conversational act being performed. For example, the speaker by performing a **Request** act likely to express his intention of having a task done; while a **Confirm** act can be performed to reassert whether the task must or must not be done (Section 3.3.1).

Table 4.5 shows the data structure for representing different types of conversational acts and the underlying intentions. The **sender** is the one who performs the act, which is towards the **receiver**. In the case of this conversational act being the response to another act, the **inResponseTo** field should be set appropriately. A special conversational act type, **Graphical**, has been introduced to capture the user's actions on the graphical interface. The interaction between the Dialogue Manager and other task assistants is also modelled as performing conversational acts. However, their interaction involves only **Request** and **Respond** acts.

In the case of the SPA, conversational acts are performed to either request for a domain task to be done, to return the results to the user, to ask for

<i>F e d</i>	<i>F e d T e</i>	<i>P b e V a e - C e</i>
id	String	the unique id of this act
sender	String	User, DialogueManager, E-mailer, Calendar
receiver	String	User, DialogueManager, E-mailer, Calendar
inResponseTo	String	the id of the conversational act to which this one is to respond to
type	ActType	Request, Respond, Request-Clarify, Clarify, Request-Confirm, Confirm, Greet, Advertise, Ack, Cancel, Graphical
content	ActContent	the associated intention

Table 4.5: Data Structure for Representing Conversational Acts

clarification, to clarify or simply to greet the user. Thus, the `content` field of the `ConversationalAct` data type is modelled to represent either a task request, a task response, a clarification request or a simple proposition. The possible types of the conversational act's contents are given in Table 4.6.

Table 4.7 shows the data fields that constitute the domain task's data structure. Thus, as in Table 4.7a, each `TaskRequest` contains information about the domain, the type and the objects of the task and finally whether or not a confirmation is required. For example, if the request was a search for messages from Paul Compton, the corresponding `TaskRequest` represented in XML is as follows:

```
<object type="TaskRequest">
  <field name="domain">EMAIL</field>
  <field name="type">SEARCH</field>
  <field name="field"></field>
  <field name="objects"><list>
    <object type="TaskObject">
      <field name="type">MAIL</field>
      <list><object type="Condition">
        <field name="attName">From</field>
        <field name="function">contains</field>
        <field name="attValue">Paul Compton</field>
      </object></list>
    </object></list>
  </field>
</object>
```

<i>Ac T e</i>	<i>Ac C e</i>	
	<i>T e</i>	<i>C e</i>
Greet	SemanticStructure	a proposition represents the semantics of an utterance
Request	TaskRequest	represents a domain task
Respond	TaskResponse	represents the results of a task
Notify	TaskResponse	represents a notification
Request-Clarify	TaskRequest	represents a task with fields marked to be clarified
Clarify	TaskRequest	represents a clarified task
Request-Confirm	TaskRequest	represents the task that requires confirmation
Confirm	SemanticStructure	a proposition represents either agreement or disagreement
Cancel	<i>e</i>	to cancel the current task and thus close the current discourse segment
Advertise	AdvertiseRequest	contains domain-specific vocabulary and other domain-specific knowledge
Ack	<i>e</i>	to acknowledge the task results and thus close the current discourse segment
Graphical	GraphicalAction	represents an action on the interface

Table 4.6: Conversational Acts and the Corresponding Act Contents

Discourse History

The *discourse history* is used for maintaining the conversational context such as information about the current and past dialogue states, which is represented as a hierarchy of discourse segments and conversational acts. Each discourse segment consists of the conversational acts contributed to that segment and has a segment purpose, which is taken to be equivalent to the user's intention (or requested task) for that segment. The Dialogue Manager agent makes use of information in the discourse history such as the user's and the system's last conversational acts, etc., in interpreting the user's current request.

<i>F e d</i>	<i>F e d T e</i>	<i>P o s s i b l e V a l u e - C o n d i t i o n</i>
domain	String	EMAIL, CALENDAR, GENERAL
type	String	type of task such as SEARCH, DELETE, CREATE, ARCHIVE, REPLY, etc.
confirmRequired	String	an optional string specifying the question to be asked the user for confirmation before performing this task
field	String	attribute to be found such as the arrival time of a message
objects	TaskObject []	an array of task objects description

(a) TaskRequest Data Type

<i>F e d</i>	<i>F e d T e</i>	<i>P o s s i b l e V a l u e - C o n d i t i o n</i>
type	String	MAIL, FOLDER, PERSON, APPOINTMENT, etc.
conds	Condition []	general conditions applicable to this object which is not domain-specific
objConds	Condition []	domain-specific conditions such as from contains "John"

(b) TaskObject Data Type

<i>F e d</i>	<i>F e d T e</i>	<i>P o s s i b l e V a l u e - C o n d i t i o n</i>
attName	String	object attribute name such as SENDER, SUBJECT, ARRIVAL_TIME, etc.
function	String	e.g. "contains", "equal"
attValue	String	object attribute value

(c) Condition Data Type

Table 4.7: Data Structure for Representing Domain Task

Discourse Segment Purpose

Utterances in a dialogue are locally related within their segments. Therefore, interpreting the user's current utterance requires the knowledge of the previous utterances in the segment and their underlying conversational acts. For exam-

ple, suppose the user's last act was of type **Request** and the Dialogue Manager's last act was a **Request-Clarify**. If the user's current utterance is a noun phrase then it is most likely that its associated conversational act is of type **Clarify**. However, if the user's current utterance is a question, the user may have cancelled his/her request and started a new segment requesting a new task (this can be determined from the semantic analysis of the utterance). In other words, the conversational act associated with the current utterance should be **Request**. Moreover, identifying the dialogue structure is also essential for resolving references such as anaphora and temporal adverbs, which is necessary for interpreting the user's utterances. The Dialogue Manager maintains information about all previously performed conversational acts in the *discourse history*.

Discourse Structure

A discourse is taken to be composed of a hierarchy of discourse segments, in which a number of segments may currently be "open" (commenced but not completed). Moreover, Grosz and Sidner [1986] have distinguished two structural relations between successive discourse segments: *dominance* and *satisfaction-precedence*. A discourse segment DS1 is considered as contributing to another discourse segment DS2 if its discourse intention DSP1 is intended to provide part of the satisfaction of the other, DSP2. Conversely, DS2 is said to dominate DS1. Moreover, DSP1 is said to satisfaction-precede DSP2 if DSP1 must be satisfied before DSP2. For our application domain, recognizing this fine-grained distinction would be computationally inefficient and unnecessary. Instead, we classify a new segment to be either the *sibling* or *sub-segment* of a previously commenced segment. A segment DS2 is a sub-segment of another segment DS1 if the correct interpretation of DS2, according to the user's intention, must require the context provided by DS1 (and implicitly other segments that contain DS1), and in addition, this contextual information must be sufficient for the correct interpretation of DS2. Otherwise, DS2 is a sibling of DS1. This definition enables the segment relations to be computed by the Dialogue Manager in the course of dialogue processing.

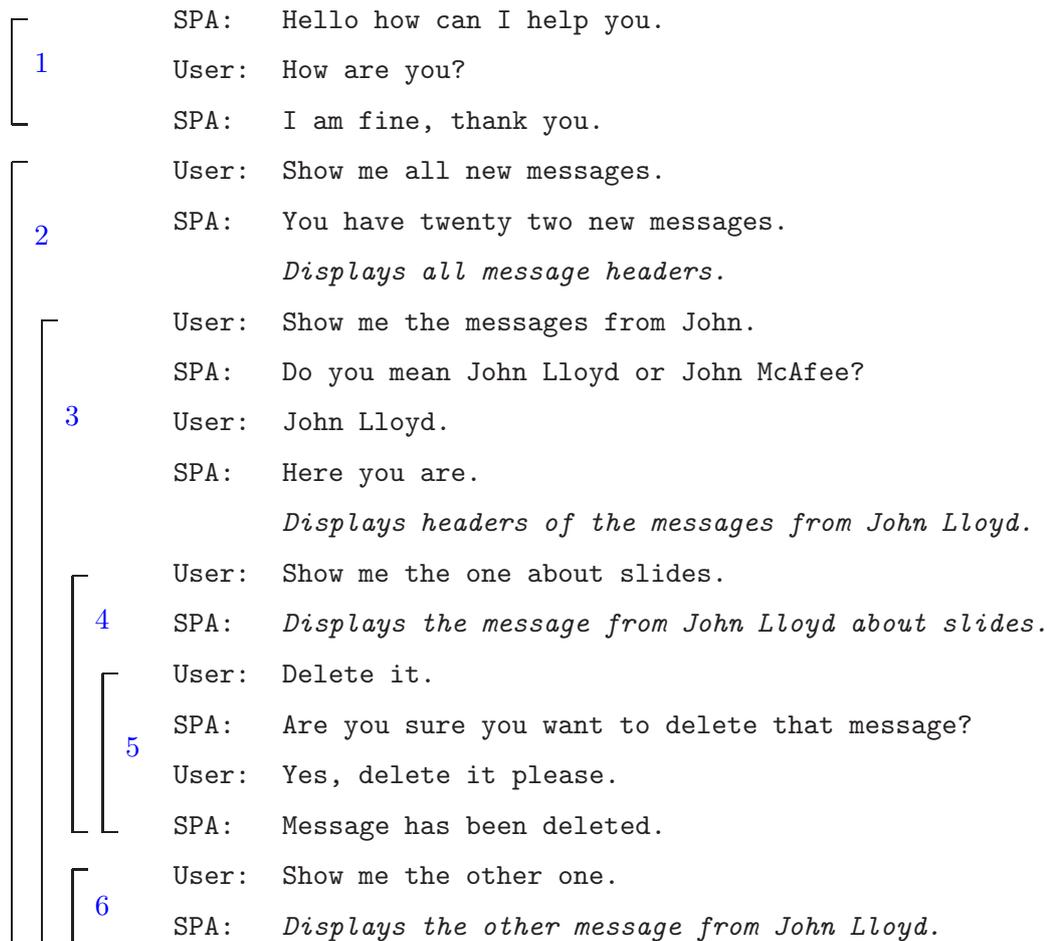


Figure 4.6: Discourse Segment Hierarchy

Taking the example in Figure 4.6, assuming that the segments have already been determined, segment (5) is a sub-segment of segment (4) because its intended interpretation requires co-reference of the new message from John Lloyd about slides, in the context set by segment (4). In contrast, segment (6) is not a sub-segment but a sibling, of segment (4). The information provided by segment (4), i.e. the message from John Lloyd about slides, is needed but not sufficient for the interpretation of “the other message” in segment (6). What must be required are the entities introduced in the context of segment (3), i.e. the two new messages from John Lloyd. Therefore segment (6) is a sub-segment of segment (3), but not of segment (4). In this example, segments (2), (3) and (6) are open while other segments have completed.

In our dialogue model, the *discourse history* contains the conversational acts performed by the user, the Dialogue Manager and other task assistants. These acts form a hierarchy of discourse segments, each having a discourse segment purpose representing the user's intention when engaging in the segment. Figure 4.6 illustrates how utterances in a dialogue constitute the hierarchy of discourse segments. Note that in this example we have ignored the interaction between the Dialogue Manager and the other task assistants, and showed only the natural language utterances exchanged between the Dialogue Manager and the user. Discourse segment (1) has the user's intention of greeting the SPA. Discourse segment (2) involves the user requesting all new e-mail messages. Furthermore, it contains a sub-segment, segment (3), because segment (3) uses the contextual information, the new messages, supplied by segment (2). Segment (3) has the user's intention of searching for e-mail from John Lloyd among the new messages. This discourse segment involves a sub-dialogue for clarification of John. However, for the purpose of modelling the dialogues in applications such as the SPA, we do not consider this sub-dialogue a sub-segment with a segment intention but rather a part of segment (3). Both segment (4) and (6) are sub-segment of segment (3) because each involves a message from John Lloyd, thus can only take place as part of the parent segment, segment (3). They are also sub-segment of segment (2). Similarly, segment (4) contains segment (5) as a sub-segment.

In many cases, it may not be possible to recognize the user's intention from the first utterance of the discourse segment (e.g. discourse segment (3) in Figure 4.6). It is, however, required that a sub-dialogue be opened requesting for additional information or clarification. We consider the sub-dialogue a part of the current discourse segment, not a sub-segment, because it relates to the same user intention (i.e. discourse segment intention). As the Dialogue Manager processes the user's current utterance, the intention that has been partially recognized is kept in the corresponding discourse segment in the *discourse history* and gradually revised until the recognition succeeds.

The Dialogue Manager also maintains in each discourse segment a stack representing the hierarchy of the conversational acts and sub-segments that constitute that segment. Figure 4.7 illustrates the discourse stack of segment (4) of the example given in Figure 4.6. Modelling the hierarchical structure of the discourse segment stacks helps in resolving references in the user’s utterances. Take the case of discourse segment (6), in which the user asks for “*the other one*”. The previous segments, segment (5) and (4), involve the message from John Lloyd about slides, which cannot contain the referenced object. However, the centring objects of segment (3) are two messages from John Lloyd. Thus, by considering the relationship among these segments, the Dialogue Manager knows that the phrase “*the other one*” can be resolved by considering the remaining of the two e-mail messages that have been mentioned in segment (3).

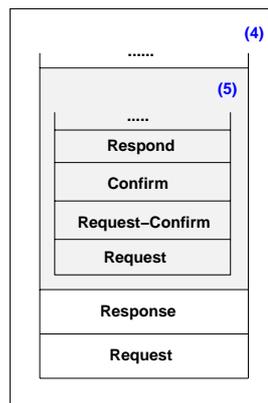


Figure 4.7: Example Discourse Stack

Recognition of New Discourse Segments

The recognition of a new discourse segment is done by checking the type of the user’s current conversational act and the Dialogue Manager’s last conversational act. As in our dialogue model, each discourse segment purpose corresponds to a domain task, a new segment most likely starts with a **Request** act of the user. Also, a **Notify** act performed by the Dialogue Manager also creates a new segment. The end of a discourse segment is either a **Respond** act from the Dialogue Manager, or an **Ack** act or a **Cancel** act by the user.

Recognition of Segment Relations

As discussed earlier, in our dialogue model, a new discourse segment is either a sibling or a sub-segment of the currently open segment. These two types of relations are identified by comparing the objects in focus of the two segments, which is based on the centring theory by Grosz, Weinstein and Joshi [1995]. They assert that certain entities or objects in an utterance are more central than others. Consequently, there is a close relationship among the focus of attention, the choice of referring expression and the coherence of utterances within a discourse. There are two levels of discourse focus: global and immediate. The centring objects of the current utterance are the immediate focus of the current discourse segment. In addition, there are other objects which are focused at the global level because they have been mentioned previously in the discourse. In our application domain, we propose that each discourse segment has a centring set of objects and that identifying the relation between two segments can be done by comparing these sets. Specifically, if the centring set of the new segment DS2 is a subset (or is the same) of the currently open segment DS1, then DS2 is a sub-segment of DS1. Otherwise, the two segments are siblings. In addition, the commence of DS2 immediately closes DS1. The example given in Table 4.8 illustrates this point.

Salient List

In the case of the SPA, although the user is not expected to use complex types of references, the use of anaphoric references and time deictic expressions is likely for the e-mail and calendar management domain. It is necessary that the Dialogue Manager is capable of resolving references of these types in order to analyse the user's utterances and recognize their associated intentions. Anaphora is the use of certain words to refer back to other referring expressions mentioned in the same or other utterances. A typical example of anaphora is the use of pronouns while time deixis is the use of expressions that are relative to a temporal reference point, such as "tomorrow", "Monday next week", etc.

	Dialogue Utterances	Centring Objects	Segment Relation
1	Show me new e-mail from Paul Compton? <i>You have one new message from Paul Compton about workshop slides</i>	message from Paul Compton	not applicable
2	Do I have any appointments with Paul tomorrow? <i>You have two appointments with Paul Compton tomorrow. One at 10am about AI workshop and the other at 3pm about agents course.</i>	two appointments with Paul Compton	sibling of 1
3	Cancel the appointment at 2pm. <i>Sorry which appointment do you want to cancel?</i> The one at 3pm. <i>Appointment has been deleted.</i>	appointment with Paul Compton at 3pm	sub-segment of 2
4	Show me the appointment at 10am. <i>Here you are.</i> Thanks.	appointment with Paul Compton at 10am	sibling of 3 and sub-segment of 2

Table 4.8: Example Relation between Dialogue Segments

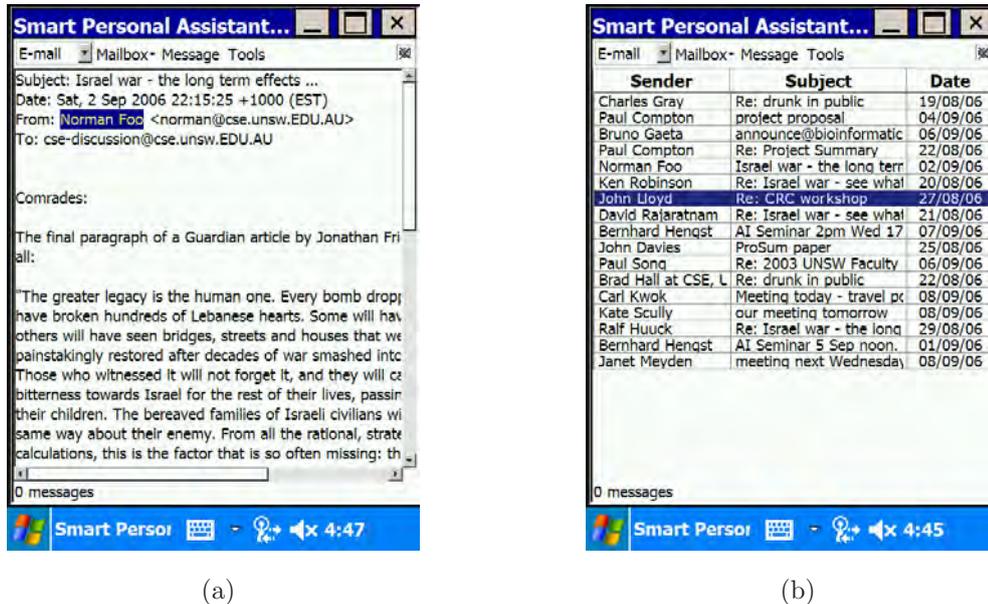
These temporal phrases are very likely to be used when scheduling activities or creating appointments.

The *salient list* of the Dialogue Manager agent is a data structure for tracking objects that have been mentioned previously during the course of conversation, and is used in interpreting references such as anaphora and temporal expressions. The *salient list* can also be used for generating context-sensitive natural language responses. For dialogue applications such as the SPA, the Dialogue Manager needs to keep track of only some kinds of discourse entities for reference resolution. For instance, in the e-mail management domain, the entities or objects of interest can be the e-mail messages, the mail boxes, the senders

and receivers of the messages and the arrival times of the e-mail messages. The user might use pronouns to refer to people or topics, e.g. for describing the content of an e-mail message, etc. In addition, the user may also use phrases to specify search conditions or people's e-mail addresses, etc. Thus those phrases should also be considered important discourse entities. Similarly, for the calendar management domain, person names, appointments and to-do tasks must be maintained. In this domain, temporal phrases such as “*tomorrow*”, “*3 pm the next day*”, etc. are used very often, hence the *salient list* must also keep track of the most recent temporal reference point.

In the SPA application, the user interacts with the system through dialogue as well as the graphical interface on the device. Hence objects involved in the interaction on the device are also kept in the *salient list* so that later references to these objects can be resolved. Figure 4.8 shows two possible examples of how objects on the interface could be referenced during a dialogue. In the first example, the user highlights a person's name on the GUI, and then asks for e-mail messages from him. Thus, the pronoun “*him*” must be resolved to “*Norman Foo*”. Similarly, in the second example, the user selects a message on the interface before requesting the system to display its content.

In addition, the user may use another type of reference known technically as *homophoric reference*, to refer to objects or people presumed to be of common knowledge in a particular social context. For instance, in the request “*Notify me if I have new e-mail from the head of school*”, the user has assumed that the system understands who the head of school is because there is only one possibility given the user's social context. This kind of information could be learned by asking the user and stored as a user preference for later processing. However, such information could also be pre-defined as a part of the domain knowledge. Currently in the SPA, the Dialogue Manager has a plan for resolving references using person names but the capability to learn and/or resolve common knowledge has not been implemented.



User Show other messages from him.

SPA *Shows other messages from Norman Foo.*

User Show me this message.

SPA *Displays content of the selected message.*

Figure 4.8: References to Objects on the GUI

Linguistic Model of the Salient List

The model of the *salient list* is based on work by Strube [Strube, 1998], which is inspired by the centring theory by Grosz, Weinstein and Joshi [1995]. The *salient list* maintains references to discourse entities that are e-mail and calendar objects. These include e-mail messages, folders, names (of contacts), appointments, time and key text phrases. In the *salient list*, these discourse entities are organized as a ranked list according to their information status, i.e. their expression types. Entities with higher ranking are more likely to be referenced in succeeding utterances. The method for determining the ranking has been given in Figure 2.1 (Section 2.1.3). A concrete description of the expression types is shown in Table 4.9. Readers are referred to the work of Prince [1981] for more detailed explanation. The *inferables* class was not used in the implementation of the SPA because its use is not required for reference resolution in our application domain.

Expression Type (Abbreviation)		Description
NEW	<i>b a d- e-</i> (BN)	Newly introduced discourse entities such as the car in the utterance: “ <i>I’ e . . . b . . . a ca</i> ”. However, people who are mentioned using their proper names such as John Lloyd in: “ <i>I - . . ee J . . L d . . F da</i> ” are considered <i>ed b a d- e-</i> entities instead.
	<i>e . . ed</i> (E)	<i>B a d- e-</i> entities after being referenced in subsequent utterances. For example, John Lloyd (as “ ”) in “ <i>I - . . ee J . . L d . . F da . N . . e . I a e e- e- a . . .</i> ”.
OLD	<i>ed b a d- e-</i> (U)	People with proper names just after being mentioned in the dialogue, such as John Lloyd in “ <i>I - . . ee J . . L d . . F da</i> ”.
MED	<i>a c . ed</i> <i>b a d- e-</i> (BN-A)	New discourse entities that are mediated by one of the other <i>e . . ed</i> or <i>ed</i> entities because of their dependence. An example is “ <i>. . e- a add e</i> ” in “ <i>I - a . . c . ac J . . . e . e e - ee . W a . . . e- a add e ?</i> ”.
	<i>e ab e</i> (I)	Discourse entities such that the speaker assumes the hearer can infer them. For example, in “ <i>Y a e . . . ee . . - a 10a . T e ca . . . ffice</i> ”, the location is implicitly assumed by both the speaker and the hearer to be the location of the appointment.
	<i>c . a</i> <i>e ab e</i> (CI)	Discourse entities contained within an inferred entity. For instance, the entities that are referred in the phrase “ <i>O e . . ee e a e</i> ” of “ <i>Y a e ee e- e- a . . O e . . ee e a e . . J</i> ”.

Table 4.9: Expression Types of Discourse Entities

Anaphora resolution is performed by searching the *salient list* from left to right according to the rank order. The objects in the referring expression are checked against discourse entities in the *salient list* until a matched type has been found. After that, the *salient list* is updated accordingly, which requires re-positioning some elements of the list to conform with the ranking criteria. In addition, brand-new objects mentioned in the dialogue are immediately inserted into the *salient list* at their determined positions. If several objects of a same type are mentioned together as a set, e.g. “*You have three new messages from John Lloyd*”, these three messages are grouped into a set (of type e-mail) before being stored as a single entity in the *salient list*. Considering the example dialogue in Figure 4.9, the set of three e-mail objects is kept in the *salient list* as a result of processing the user’s first request. Although there may be other types of discourse entities (e.g. the Seminars folder) currently in the *salient list*, only the e-mail set entity is matched and used for resolving the reference of “the second message” in the user’s second request.

User Show me messages from John in the Seminars folder?
SPA There are three messages from John Lloyd in the Seminars folder.
 Displays the three message headers.
User Show me the second message.
SPA Here you are.
 Displays content of the second message.

Figure 4.9: Dialogue Illustrating Reference Resolution Using the Salient List

4.3 Dialogue Processing

Inputs to the dialogue control process are either the syntactic frames of the user’s utterances or the descriptions of the user’s GUI actions. The classification of the input into a task domain is considered a part of the syntactic analysis sub-process, however, will be discussed in this section as this is done by the Di-

alogue Manager agent. In the main part of this section, we describe a typical course of dialogue processing performed by the Dialogue Manager, which can be divided into different sub-processes: analysing the semantics of the utterance, identifying the user's intention, processing the requested task and generating system responses. Figure 4.10 illustrates a part of the dialogue processing's control flow, which is derived automatically as the Dialogue Manager agent selects and executes its dialogue plans according to the context. In the diagram, the curved rectangles are the agent's plans. Lines of type (1) denote access from and update to the Dialogue Manager's beliefs. Type (2) lines are for illustrating flows of successful plan execution while type (3) lines indicate the dialogue processing flow as the result of plan execution failures. The same notation will be used in other diagrams throughout the remainder of this section. All dialogue examples should be referred back to the complete scenario in Figure 4.2.

The Dialogue Manager agent has been implemented as a JACK agent, which has a set of modular plans, as described previously. Figure 4.11 illustrates how to specify a plan in JACK. The body of the plan defines the sequence of actions that the agent must do in handling the input event. If the execution of the plan fails at some point, the plan is immediately terminated. Nevertheless, if the `Fail` reasoning method has been specified, the agent will execute this reasoning method before terminating the plan execution. The context of the plan is important for deciding which plan to execute: for example, in our domain we might have two plans for displaying e-mail contents, one for use in the context that the user is on the Desktop and another for when the user is on a PDA, which might summarize the message first before displaying it.

4.3.1 Domain Classification

A new user input is either a spoken utterance or a graphical action. Thus the input to the *Domain Classification* plan can be a syntactic frame of the utterance (as processed by the Partial Parser) or a description of the graphical

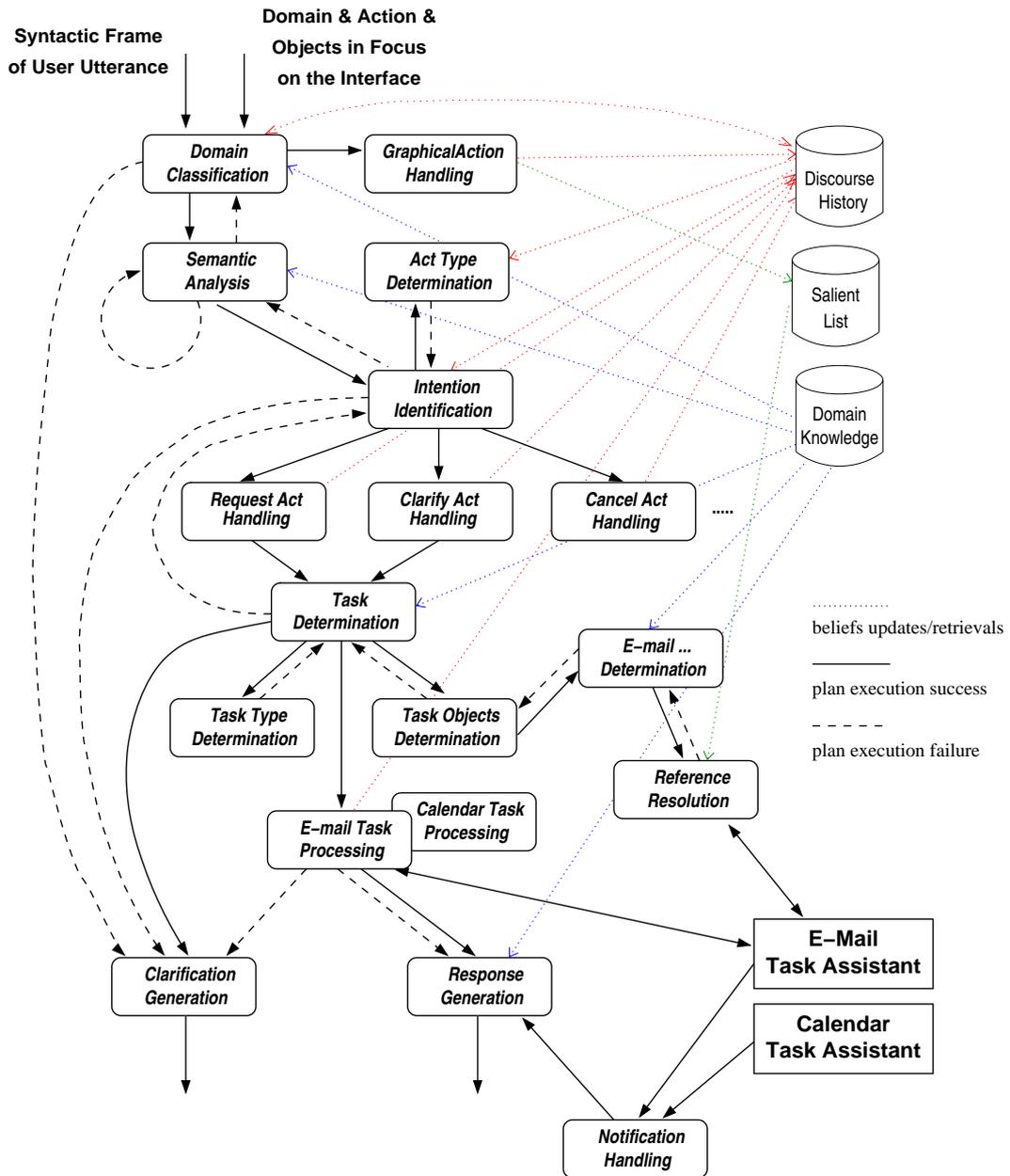


Figure 4.10: Illustration of Dialogue Control Flow

action. If the input is an action description, an internal event will trigger the execution of the *Graphical Action Handling* plan for further processing. Otherwise, the Dialogue Manager agent performs steps in the *Domain Classification* plan to calculate a ranked list of domains that the user's current utterance is likely to belong to. Some domain-specific knowledge and information of the conversational context is used for the classification.

<i>Handle Event:</i>	internal or external event that this plan handles
<i>Post Event:</i>	internal event that this plan can raise, which must be handled by itself
<i>Send Event:</i>	external event that this plan can raise, which is to be handled by another agent
<i>Relevance:</i>	conditions to check if this plan is a relevant option to handle the input event
<i>Context:</i>	conditions to check if this plan is applicable for handling the input event
<i>Body:</i>	sequence of actions that the agent to carry out, which can be one of the following: <ul style="list-style-type: none"> - perform an atomic action - post an internal event - send an external event - execute another reasoning method
<i>Reasoning Method:</i>	sequence of actions that the agent to carry out, which can be one of the following: <ul style="list-style-type: none"> - perform an atomic action - post an internal event - send an external event - execute another reasoning method
<i>Pass:</i>	actions to carry out after successful execution of the plan
<i>Fail:</i>	actions to carry out if the plan execution has failed

Figure 4.11: JACK Plan Specification Schema

A ranked list of all supported domains is calculated for the input utterance using the domain-specific keywords kept in the Dialogue Manager's domain knowledge. There is a set of keywords for each supported domain. Each keyword has a confidence value between 0 and 100, which indicates how likely *a priori* the utterance belongs to a particular domain given the occurrence of that keyword. A word can appear as the keyword in different domains though its confidence values in these domains are unlikely to be the same. In the SPA, the

confidence values are pre-specified as domain-specific information. It would be possible to use statistical methods to compute these values using training data. However, in our implementation of the SPA, we manually pre-define the values according to the following simple rules:

- If a word can appear in utterances of both e-mail and calendar domains, it is given a value of 50. Examples are verbs such as “move”, “delete”, “show”, and nouns such as “details”, etc.
- If a word is absolutely a keyword of one domain but not the other, it is given a value of 100. For example, words such as “schedule” are given a value of 100 in the calendar domain while words such as “archive” are given a value of 100 in the e-mail domain.
- If a word is most likely to appear in utterances of one domain, but can also be a keyword in another domain though less likely, it is given a value of 80. The confidence value of that word in the other domain will be 20. For example, the occurrence of the noun “seminar” most likely indicates the calendar domain, such as in *“I am going to have a seminar this Friday”*. However, the same word can also appear in an utterance of the e-mail domain such as *“Show me the messages about seminars”*.

The ranking value for a domain is the sum of the ranking values of all domain-specific keywords that appear in the utterance. The larger the ranking value is, the more likely that the requested task belongs to the domain. The ranked list is maintained in the agent’s *discourse history* but a domain with highest ranking value is selected first. If the ranking values are the same, the currently active domain (domain of the last conversational act in the dialogue) is given higher priority. The domain knowledge is indexed so that (as the result of selecting a domain) the Dialogue Manager will automatically use the appropriate domain knowledge in later stages of the dialogue processing. However, if the dialogue processing fails at some later point (e.g. identification of the user’s intention

fails), it is possible for the Dialogue Manager to roll back and select the next highly ranked domain from the ranked list.

Considering utterance (1), “*Is there any e-mail from Paul Compton*”, in our dialogue example (Figure 4.2), there are no keywords in the calendar domain but one keyword in the e-mail domain, i.e. “e-mail”, which has a confidence value of 100. Thus the utterance can be classified to the e-mail domain. In the next utterance, “*Show me the one about proposal*”, the word “show” appears as a keyword in both domains with the same confidence value of 50. Thus the ranking values are the same for both domains. However, as the previous utterance belongs to the e-mail domain, this utterance must also be classified to the e-mail domain. In contrast, if the utterance was “*Show me the one about appointment*”, the ranking value for the calendar domain would be higher, which would first indicate that the utterance is a calendar request. Nevertheless, as the utterance contains an anaphoric reference “the one”, this will make the dialogue processing fail at a later stage, semantic analysis, and so the Dialogue Manager will be able to roll back and attempt to reprocess the utterance in the e-mail domain.

Figure 4.12 shows one part of Figure 4.10, displaying only the flows of data and execution from and to the *Domain Classification* plan. Note again that the dotted lines (1) indicate accesses from and updates to the Dialogue Manager’s beliefs. Solid lines (2) are flows of successful plan execution while dashed lines (3) denote dialogue processing flows as the result of plan execution failures. If the classification succeeds, the execution of the *Semantic Analysis* plan is triggered, in which the Dialogue Manager attempts to formulate a semantic representation of the user’s utterance using specific knowledge of the classified domain. If the *Semantic Analysis* plan fails, the Dialogue Manager selects the next ranked domain and re-execute the *Semantic Analysis* plan. If the input is a GUI action, an internal event is posted triggering the execution of the *Graphical Action Handling* plan.

4.3.2 Semantic Analysis

Only if the user's utterance has been classified into a domain, the *Semantic Analysis* plan can be used to formulate a semantic representation of the utterance using the general linguistic knowledge encoded in the Dialogue Manager's beliefs and other domain-specific knowledge. This domain-specific knowledge is selected automatically according to the classified domain. The semantic representation of the user's utterance is then used for identifying the underlying intention, which can be a new request or a clarification to the system's last question. As we have discussed previously, if the execution of the *Semantic Analysis* plan fails, the next possible domain will be selected and the Dialogue Manager will attempt to re-do the semantic analysis using the newly selected domain knowledge (Figure 4.12).

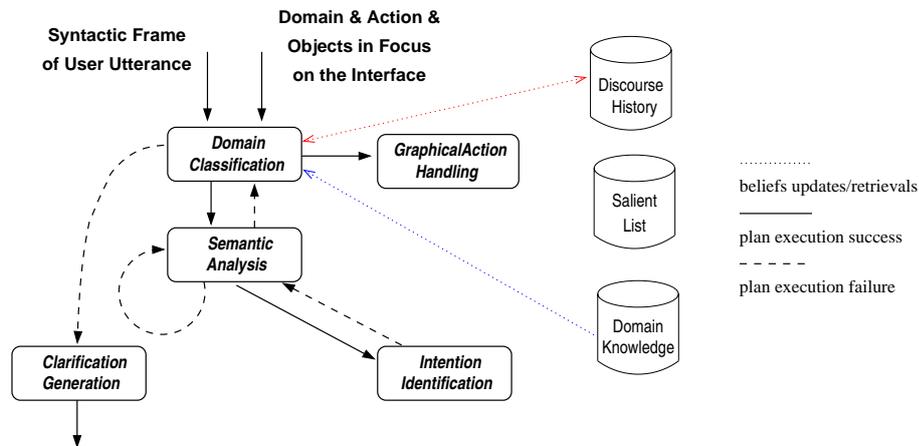


Figure 4.12: Dialogue Control for Domain Classification and Semantic Analysis

The input to the Dialogue Manager is the syntactic structure of the user's original utterance (e.g. Table 4.3). Using the *Semantic Analysis* plan, the Dialogue Manager attempts to convert this syntactic structure into a semantic structure, which is either a predicate proposition, a quantifier proposition or an operator proposition. These three types of propositions can be described as follows and the representation of the semantic structure is formally defined in Figure 4.13. An example semantic representation is given in Figure 4.14.

- *Predicate Proposition:* consists of a predicate and a set of terms. Each term can be a text phrase; a constant term representing a pronoun; or a variable term that is linked to another quantifier proposition. For example, if the utterance is “*Show me new messages*”, then the predicate is **show** and the two terms are the constant term **me** and a variable term corresponding to the phrase **new messages**.
- *Operator Proposition:* consists of an operator and a set of propositions. The operator is used to represent either (i) the tense of a proposition if only one single proposition is included; or (ii) a conjunction or disjunction of the included set of propositions.
- *Quantifier Proposition:* consists of a quantifier, a variable and a set of propositions. The set of propositions represent conditions or characteristics of the variable term such as the type of the associated object, its ordinal position within a list, etc. For example, a phrase such as “*new messages*” can be represented as a quantifier proposition, which contains two other predicate propositions: one representing the e-mail message type of the objects and the other representing the condition of these e-mail objects, i.e. being new.

At this stage, the semantic representation of the utterance is not yet final as it must be refined further using domain-level plans to allow the Dialogue Manager to fully recognize the requested task. For example, a phrase such as “*messages from Paul Compton*” can only be formulated as a variable term, representing a set of e-mail objects with some additional condition. Interpreting this additional condition, i.e. the attribute description “*from Paul Compton*”, requires detailed specific-knowledge of the e-mail management domain, thus will be further processed using the domain-level dialogue plans, e.g. *E-mail Determination* plan. Anaphoric references such as pronouns and definite noun phrases are also not resolved at this stage. Nevertheless, the resulting semantic structure is sufficient to determine the user’s conversational act and partially understand

```

SemanticStructure : UtteranceType + Proposition
Proposition      : Operator      + Proposition[]
                  Predicate      + Proposition  + Term[]
                  Quantifier     + Variable     + Proposition[]
Operator         : AND | NOT | OR | PRESENT | PAST | FUTURE
Predicate       : Have | Be | BeingNew | BeingFine | ObjectType ...
Quantifier      : MOST | MANY | ALL | ANY | THE | A | ...
Term            : Constant
                  : Variable
                  : Operator      + Term[]
Constant       : You | I | Mary | John | ...
Variable       : x | y | z | ...

```

Figure 4.13: Semantic Structure Representation

its associated intention. Taking the example utterance (3), “*Show me the one about proposal*”, its semantic structure is represented in Figure 4.14.

```

IMPERATIVE
AND PRESENT Show("You", "I", x)
  THE x: AND ObjectType(x, ONE, SINGULAR)
         ObjectProperty(x, ABOUT, "proposal")

```

Figure 4.14: Example Semantic Structure

4.3.3 Intention Identification

In order to maintain a coherent dialogue with the user, the Dialogue Manager needs to understand the user’s intentions when speaking an utterance or performing an action on the interface. Not all user interactions are requesting tasks but some are for clarifying ambiguities or providing more information. Therefore recognizing the user’s intention involves, firstly, identifying what kind of conversational act is being performed, and then, depending on the type of the

user’s conversational act, determining the content of the act. More specifically, if the user’s conversational act is a **Request**, the requested task must be recognized. Otherwise, if the user is performing a **Clarify** or a **Confirm** act, the Dialogue Manager must determine what is being clarified or confirmed. This requires analysing the semantic structure of the user’s utterance and accessing the conversational context maintained in the agent’s *discourse history*.

Figure 4.15 illustrates a part of the course of dialogue processing, identifying the user’s intention when speaking an utterance. Again, the dotted lines (1) indicate accesses from and updates to the Dialogue Manager’s beliefs. Solid lines (2) are flows of successful plan execution while dashed lines (3) denote dialogue processing flows as the result of plan execution failures.

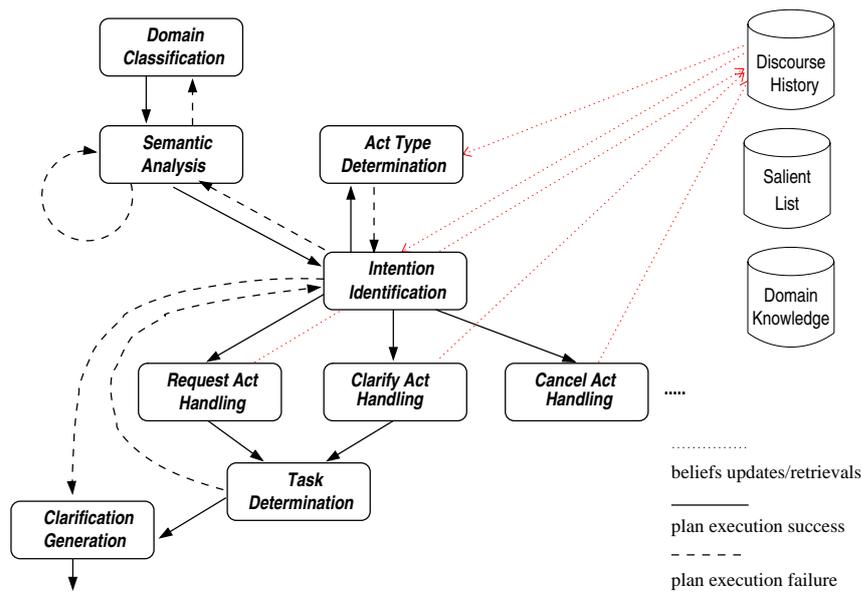


Figure 4.15: Dialogue Control Flow for Intention Identification

In executing the *Intention Identification* plan, the Dialogue Manager first triggers the execution of the *Act Type Determination* plan for handling the sub-task of determining the user’s conversational act. This is done by considering the syntactic and semantic features of the user’s utterance and the previous conversational acts performed by the user and the Dialogue Manager. The result is an ordered list of possible act types and the top result will be chosen

for further processing. There is a plan for handling each conversational act type such as *Request Act Handling*, *Cancel Act Handling*, etc.

If the execution of the *Act Type Determination* fails, causing the failure of the *Intention Identification* plan, the Dialogue Manager would select the next ranked domain from the domain classification result and continue processing the current utterance by re-executing the *Semantic Analysis* plan. For example, consider utterances (5–9) in our example dialogue (Figure 4.2):

- 5 User I need to make an appointment with him.
 6 SPA Please specify date and time of the appointment?
 7 User Monday at 11 am.
 8 SPA What is the appointment title?
 9 User Project archive.

In the beginning, utterance (9) is classified to the e-mail domain because it contains the keyword “archive”². However, in the previous utterance (8), the Dialogue Manager was performing a *Request-Clarify* act in a different domain, i.e. the calendar domain. Moreover, the user’s current utterance (9) is short, having less than three words. Hence it is more likely that the user’s current conversational act is a *Clarify* act in responding to the system’s *Request-Clarify* act, and thus must belong to the calendar domain. As a result, the Dialogue Manager drops the execution of the *Act Type Determination* plan, adjusts the classified domain and re-does the semantic analysis of the utterance. Figure 4.16 gives the pseudo-code illustrating the execution of the *Intention Identification* plan. *Body* and *DetermineAct* are reasoning methods. The `return true` (or `return false`) statement indicates the reasoning method has succeeded (or failed).

Once the Dialogue Manager has selected an *Act* from the *PossibleActs* set as the user’s current conversational act, the execution of a corresponding plan

²The shallow partial parser has only a limited vocabulary to be computationally efficient, to reduce the effects of speech recognition errors and to avoid restricting the user’s vocabulary, so is not able to detect that the utterance is in fact just a noun phrase and the word “archive” is a noun, not a verb.

```

Handle Event: IdentifyIntentionEvent
Post Event:
    DetermineActEvent, HandleActEvent,
    ProcessTaskEvent, AnalyzeSemanticsEvent
Body:
    if (reasoning method DetermineAct failed) then
        post AnalyzeSemanticsEvent
        status = false
        return true
    for each Act in PossibleActs do
        UserAct = Act
        // only one relevant plan among RequestActHandling,
        // CancelActHandling, etc. will handle HandleActEvent
        if (subtask HandleActEvent failed) then
            UserAct = next Act
        else
            status = subtask result
            break
DetermineAct:
    if (subtask DetermineActEvent failed) then
        select next classified domain
        return false
    else
        PossibleActs = subtask result
        return true
Pass:
    if (status == true) then post ProcessTaskEvent

```

Figure 4.16: Intention Identification JACK Plan

among *Request Act Handling* plan, *Clarify Act Handling* plan, etc. is triggered immediately. If the execution of this plan fails, the next possible conversational act will be chosen for re-processing. If the execution of the plan succeeds and the plan returns a `true` status, indicating the user's request has been fully recognized, a *ProcessTaskEvent* event will be posted so that the Dialogue Manager agent will start processing the task. However, if the act handling plan has succeeded but a `false` status has been returned, the request has not been fully recognized, and so no task will be processed yet.

Conversational Act Determination and Handling

Recognition of the user’s conversational act is based on analysing the previous acts that are performed by the user and the Dialogue Manager. Moreover, it also depends on the syntactic class of the user’s utterance. The *Act Type Determination* plan uses an ordered set of heuristic rules to associate with each utterance an ordered list of possible act types. The rule order is important because it affects the order of the result list if there is more than one rule matched. The Dialogue Manager continues processing the dialogue using the result starting from top of the list, until the user’s intention can be successfully identified. Some example heuristic rules for determining the act type are shown in Table 4.10.

Rule	User’s last act	DM’s last act	Utterance type	Result
1	Request	Respond	<i>e a e</i>	Request
2	Clarify	Respond	<i>- - e</i>	Request
3	Cancel	Ack	<i>e . - e</i>	Request
4	Request	Request-Clarify	<i>dec a a</i>	Clarify
5	Request	Request-Clarify	<i>e a e</i>	Cancel
6	Request	Request-Confirm	<i>dec a a</i>	Confirm

Table 4.10: Example Heuristic Rules for Act Type Determination

When the Dialogue Manager generates an internal event indicating that the user’s conversational act has been identified, due to the execution of the agent’s BDI interpreter, the most appropriate plan for handling that act type will automatically be selected for execution. If this plan fails, i.e. the user’s intention cannot be determined, most likely because the recognized act type was not right, the Dialogue Manager selects the next matched act type and re-posts the event to try interpreting the user’s intention again. Considering the description of the defined conversational acts, the user should only perform some of the available acts: **Request**, **Clarify**, **Confirm**, **Cancel**, **Greet** or **Ack**. There is a plan

for handling each of these conversational acts. For example, if the user's act is a **Request**, the *Task Determination* plan will be selected to determine the details of the requested task. However, if the user is performing a **Cancel** act, the *Handle Cancel Act* plan is executed, which generates an acknowledgement for grounding and closes the current discourse segment. Otherwise, the *Handle Clarify Act* plan is selected in the case the user's act is **Clarify** or **Confirm**. This plan will then post a *DetermineTaskEvent* event so that as a result, the *Task Determination* plan is executed to update the segment intention accordingly. The control flow between the various plans for task determination is illustrated in Figure 4.17.

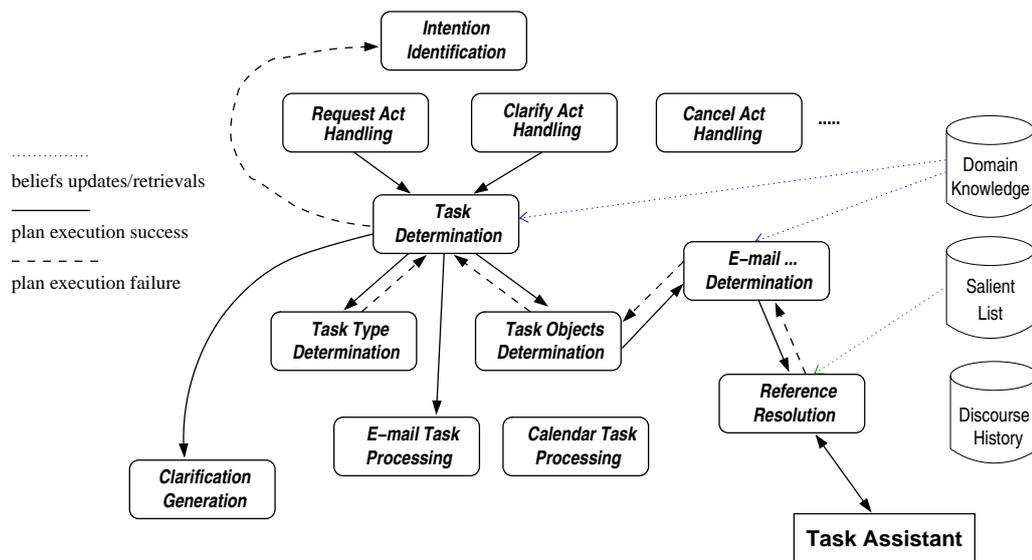


Figure 4.17: Dialogue Control Flow for Task Determination

In order to determine the user's requested task, the Dialogue Manager uses domain knowledge to identify the type of the task and the associated objects. In many cases, determining the task requires resolving references, which is done by triggering the execution of the *Reference Resolution* plan. If the user's current conversational act is a **Clarify**, the new task information obtained from processing the current utterance is merged with the existing information, which is maintained as a partially recognized intention in the *discourse history*.

The pseudo-code of the *Task Determination* plan is shown in Figure 4.18. In executing this plan, the Dialogue Manager determines or updates the user's requested task including the task type and the objects, then determines if the task has been fully recognized. If a full task description cannot be formulated, for example, due to missing or incorrect information, a *GenerateClarification-Event* event is posted, triggering the appropriate plan to generate a clarification question. For instance, in our example scenario (Figure 4.2), the user's utterance (5) suggests a calendar task of creating an appointment. The attendee has been provided, but not the time and the title of the appointment. Thus the system responds by a question, utterance (6), requesting the appointment's date and time. Similarly, in utterance (13), the user provides a non-existent folder, hence the e-mail task object cannot be identified, which causes an execution failure of the subtask triggered by the *DetermineObjectsEvent* event.

Task Type Determination

Each domain task has a task type and a set of domain-specific objects. Using available domain knowledge, the Dialogue Manager identifies the task type based on the recognition of the main predicate of the utterance's semantic structure. The mapping of predicates and task types is pre-defined in the task assistants and provided in advance to the Dialogue Manager. A special flag is included in each mapping to indicate if the task should not be done without a confirmation from the user. For example, deletion of an appointment should be taken with care (e.g. utterance (19) in Figure 4.2). Thus processing of a *Delete* task with appointment objects would trigger the execution of the *Clarification Generation* plan to request the user to confirm the deletion (e.g. utterance (20)). Nevertheless, whether a confirmation is needed depends on the user's preferences. Therefore there is a motivation for learning user preferences to support dialogue adaptation. Figure 4.19 gives the pseudo-code of the *Task Type Determination* JACK plan.

```

Handle Event:
    DetermineTaskEvent
Post Event:
    DetermineTaskTypeEvent, DetermineObjectsEvent,
    GenerateClarificationEvent
Body:
    if (subtask DetermineTaskTypeEvent failed) then
        return false
    if (subtask DetermineObjectsEvent failed) then
        status = false
        post GenerateClarificationEvent
        return true
    if (reasoning method MoreInfoRequired failed) then
        status = false
        post GenerateClarificationEvent
        return true
MoreInfoRequired:
    if (TaskType not determined) then return false
    get RequiredObjects from DomainKnowledge using TaskType
    for each TaskObject in RequiredObjects do
        if (TaskObject not determined) then
            post GenerateClarificationEvent
    return true

```

Figure 4.18: Task Determination JACK Plan

Attribute Extraction and Object Determination

Depending on the task type, certain types of objects are also required in order to perform the task. There is a set of domain-dependent plans for recognizing task objects mentioned in the user's utterance. Firstly, object attributes must be extracted from the input utterance. The current semantic structure of the utterance is further expanded using additional domain knowledge so that attributes can be extracted. Domain-specific knowledge is required because different domain objects require different types of attributes and are expressed differently in natural language. For the e-mail management domain, there is an *E-mail Determination* and a *Folder Determination* plan for extracting e-mail and folder attributes, respectively. For instance, text phrases such as "from Marc Huber" or "about workshop slides" must be recognized as expressing attributes of

```

Handle Event:
    DetermineTaskTypeEvent
Body:
    if (Act == Confirm) then
        if (reasoning method UpdateTaskType failed) then
            status = false
            set plan result = status
            return true
        else reasoning method DetermineTaskType

DetermineTaskType:
    get MainVerb from SemanticStructure
    TaskType = get MappedTask from DomainKnowledge using MainVerb
    if (TaskType not found) then
        TaskType = get DefaultTask from DomainKnowledge
    return true

UpdateTaskType:
    // user cancels current request
    if (UtteranceType is Negation) then
        ActType = Cancel
        return false
    // user confirms current request
    else if (UtteranceType is Affirmation) then
        return true
    // user may start a new request
    else
        return false

```

Figure 4.19: Task Type Determination JACK Plan

some e-mail messages, i.e. the sender is Marc Huber and the message's topic is workshop slides. Similarly, appointment attributes are extracted using the *Appointment Determination* plan. For example, attributes required for making a new appointment can include a short description, the time and place of the appointment, its category and the attendees.

Graphical Action Handling

In the SPA application, the user interacts with the system through a dialogue as well as the system interface on the device. Hence, in order to maintain the

```

Handle Event:
    DetermineObjectsEvent
Post Event:
    DetermineObjectEvent, GenerateClarificationEvent
Body:
    if (Act == Request) then
        if (reasoning method DetermineObjects failed) then
            status = false
            set plan result = status
            return true
        else
            if (reasoning method UpdateObjects failed) then
                status = false
                set plan result = status
                return true

DetermineObjects:
    get RequiredObjects from DomainKnowledge using TaskType
    for each ObjectType in RequiredObjects do
        // the relevant plan among EmailDetermination,
        // FolderDetermination, etc. plans will handle this subtask
        if (subtask DetermineObjectEvent failed) then
            post GenerateClarificationEvent
            return false

UpdateObjects:
    for each ObjectType in DeterminedObjects do
        if (update ObjectType failed) then
            post GenerateClarificationEvent
            return false

```

Figure 4.20: Task Objects Determination JACK Plan

context, objects that are involved in the interaction on the device are sent to the Dialogue Manager for updating the *salient list*. These updates are done by the *Graphical Action Handling* plan. The data description for representing the GUI actions is given in Table 4.11. Take the case of the user selecting an appointment on the interface and requests the system to cancel it: “*Cancel this appointment*”. The appointment object must have already been in the *salient list* for the Dialogue Manager to be able to resolve the reference.

<i>F e d</i>	<i>F e d T e</i>	<i>P . b e V a e - C . e</i>
domain	String	EMAIL, CALENDAR, GENERAL
action	String	SELECT, HIGHLIGHT, SHOW, SHOW_CONTENT, etc.
objects	GUIObject[]	an array of graphical objects description

(a) GUI Action Data Type

<i>F e d</i>	<i>F e d T e</i>	<i>P . b e V a e - C . e</i>
type	String	type of the object e.g. EMAIL, FOLDER, APPOINTMENT, PHRASE, etc.
value	String	the highlighted phrase or the id of the GUI object

(b) GUI Object Data Type

Table 4.11: GUI Action Data Type

Domain objects on the interface are treated as discourse entities. If they become in focus, they are placed into the *salient list*. Although this method can only support limited multimodal interaction, the e-mail and calendar domain do not require time-sensitive fusion of multimodal information. Our agent-based approach is also applicable for other application domains such as map-based directory search, etc., where multimodal interaction is essential. What is required is an effective method for multimodal fusion of inputs and interpreting this combination [Oviatt, 1999].

Reference Resolution

If there are any references to task objects, they must be resolved before the task is fully identified. In this case, an internal event is posted within an object-determination plan, which triggers the selection of the *Reference Resolution* plan for resolving the references. The type of the referenced object and the attributes of the reference must be passed on to the *Reference Resolution* plan. In addition, temporal phrases are also resolved using this plan. However, person names or

other person-related pronouns and definite noun phrases such as “*the head of school*”, etc., are resolved by the *People Determination* plan. At the moment, the plan resolves pronouns and person names using the current context. However, it is also possible that the *People Determination* plan could access an address book available at the back-end task assistants to retrieve person names that match the reference. Moreover, if any learning mechanism can be integrated, it would be possible to learn the user’s preferences in referring to people using definite noun phrases, short or nick names. This interesting learning problem is one for future work.

In executing the *Reference Resolution* plan, the Dialogue Manager resolves references to task objects by looking for matched entities in the *salient list* (c.f. Section 4.2.4). Pronouns can be resolved by matching both the object type and the number of objects if the reference is to a set of objects. However, if the reference was used to introduce a *containing inferable* entity, the Dialogue Manager is required to interact with the appropriate task assistant to identify the contained objects. An example of reference to a containing inferable entity is illustrated in the example scenario in Figure 4.2:

- 1 User Is there any e-mail from Paul Compton?
- 2 SPA You have two messages from Paul Compton. One is about project summary and one is about project proposal.
Displays the two message headers.
- 3 User Show me the one about proposal please.
- 4 SPA Here is your message about proposal.
Displays the message from Paul Compton about project proposal.

After the system displays two message headers on the interface, the set of these two messages is currently in focus, so is most likely to be referenced in the next utterances. In the next utterance, the user refers to the message about “*proposal*”, which should be one of the two messages. Thus the Dialogue Manager requests the e-mail task assistant to identify which message is about “*proposal*”.

Currently this is done by looking for keywords in the message's subject and body contents. However, it is possible that the e-mail task assistant can use the e-mail classification results.

The Dialogue Manager agent maintains a temporal reference point in the *salient list* for use in resolving temporal phrases. For instance, considering utterance (7) in our example dialogue, the user specifies the time of the new appointment to be Monday at 11am, which refers to next Monday, hence must be resolved using the current date and time as the reference point:

- 5 User I need to make an appointment with him.
6 SPA Please specify date and time of the appointment?
7 User Monday at 11 am.

Taking another example in Figure 4.21, the original time of the appointment can be used as the temporal reference point for resolving the new appointment time. By default, if the user specifies only the new day, e.g. Friday as in the example, the system interprets that the appointment should be moved to the same time on Friday. However, it would be possible that the system asks the user to suggest a time.

-
- User Do I have any appointment with John this week?
SPA You have an appointment with John Lloyd about travel on Wednesday at 3 pm.
Displays the calendar view of this Wednesday.
User Move this appointment to Friday.
SPA Are you sure you want to reschedule the appointment with John Lloyd to 3 pm this Friday?
User Yes.
SPA Appointment has been rescheduled.
-

Figure 4.21: Temporal Phrase Resolution

Clarification Request Generation and Handling

Clarification Request Generation

If the Dialogue Manager cannot determine the task type or fails to determine the task objects, an event will be generated so that the *Clarification Generation* plan is selected for execution. A *Request-Clarify* act will be performed by the Dialogue Manager in this conversational situation. Depending on what information is missing, the Dialogue Manager generates a context-sensitive question using domain-specific knowledge. The domain knowledge contains a mapping from the object type and attribute to a template for generating a clarification question. As a result, to ask for an appointment's attendee(s), a question such as "*Who are you going to meet?*" is used (e.g. in the example in Figure 4.22), while to clarify a person name for sending a message, question such as "*To whom should the message be sent?*" is used as it is more appropriate.

User I need to have a meeting tomorrow at 3 pm about ARC grant.
SPA Do you want to create that appointment?
User Yes, please.
SPA Who are you going to meet?
User Paul Compton.
SPA Appointment has been created.

Figure 4.22: Clarification Request Generation and Handling

Request-Confirm acts are very similar to *Request-Clarify* acts, but are performed by the Dialogue Manager to ask the user for a confirmation. It is required if the performance of a task must be confirmed either to avoid irreversible mistakes or because the user's desire was not certain. For instance, in the example in Figure 4.22, the user has stated a need but not explicitly requested the system to create a new appointment. A sophisticated dialogue requires that the user is asked for confirmation. Again the domain knowledge is used automatically to tailor the question according to the type of the task.

Clarification Handling

After a clarification or confirmation request is returned to the user, the Dialogue Manager updates the *discourse history*. The Dialogue Manager's conversational act is pushed onto the discourse stack of the current discourse segment. The partially recognized segment intention is also maintained here. If the user returns with a *Clarify* or *Confirm* act, the Dialogue Manager uses the conversational context to interpret the new information in the user's utterance and updates the segment intention accordingly. If the intention can be fully recognized, the corresponding task will be performed. Otherwise, additional clarification or confirmation requests will be generated.

4.3.4 Task Processing

When a user request is identified, the Dialogue Manager posts an internal event as an indication. The dialogue model has a number of plans for handling this event but only the one that matches the task domain will be selected. Tasks can be performed immediately if the necessary knowledge is available in the Dialogue Manager's internal beliefs, e.g. displaying the content of a message that has been mentioned previously in the dialogue. Otherwise, the Dialogue Manager requests more information from the relevant back-end agent or simply delegates the tasks and waits for the results. Some tasks may require sending more than one request, thus the interaction with the other task assistants is also maintained in the Dialogue Manager's *discourse history*. Figure 4.23 illustrates the control flow of plan selection and execution for processing and handling tasks.

There can be two-way information flow in the dialogue. The user can request to be notified of particular events, which may require the Dialogue Manager to register those events with other back-end agents. Upon receiving notification, the *Notification Handling* plan processes the notification and activates another plan, which generates the response to notify the user. The Dialogue Manager

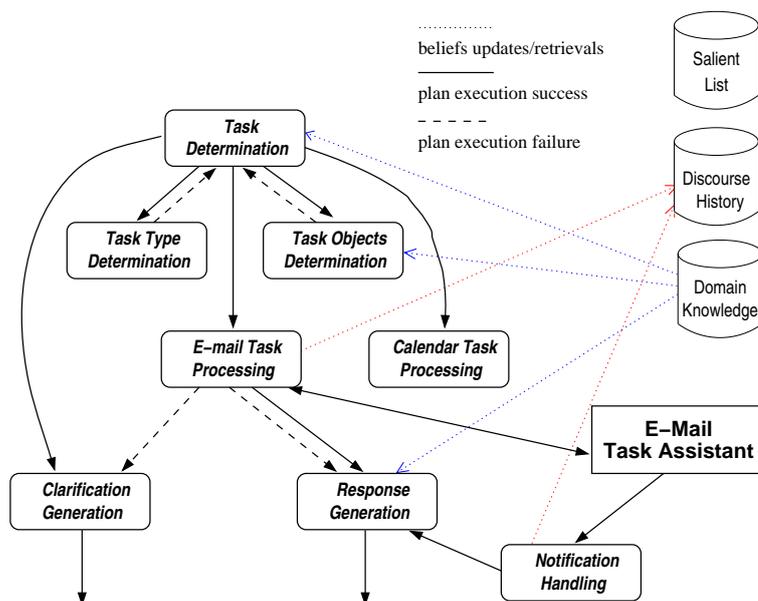


Figure 4.23: Dialogue Control Flow for Task Processing and Handling

may choose to immediately notify the user or defer the notification because prompt notifications might distract the user from other ongoing tasks. The *Notification Handling* plan can be paused and resumed later. This is another motivation for integrating learning into the dialogue model to learn when should the user be interrupted, which is possibly a future work.

As the user's task has been handled successfully, the system's *Respond* act completes the current discourse segment. Note that the *discourse history* is continuously updated during the processing of the segment. Table 4.12 illustrates the discourse stack of the segment from utterances (5) to (10) in the example scenario:

- | | | |
|----|------|---|
| 5 | User | I need to make an appointment with him. |
| 6 | SPA | Please specify date and time of the appointment? |
| 7 | User | Monday at 11 am. |
| 8 | SPA | What is the appointment title? |
| 9 | User | Project archive. |
| 10 | SPA | Appointment has been created for next Monday 11 am. |

<i>type</i>	<i>id</i>	<i>sender - receiver</i>	<i>content</i>
Request	5	User - SPA	domain=CALENDAR, type=CREATE, objects=[{type=APPOINTMENT, objConds=[attendees="Paul Compton"]}]]
Request-Clarify	6	SPA - User	domain=CALENDAR, type=CREATE, objects=[{type=APPOINTMENT, objConds=[time=?]}]]
Clarify	7	User - SPA	domain=CALENDAR, objects=[{type=APPOINTMENT, objConds=[time="Monday 11 am"]}]]
Request-Clarify	8	SPA - User	domain=CALENDAR, type=CREATE, objects=[{type=APPOINTMENT, objConds=[title=?]}]]
Clarify	9	User - SPA	domain=CALENDAR, objects=[{type=APPOINTMENT, objConds=[title="Project archive"]}]]
Request	10	SPA - Calendar	domain=CALENDAR, type=CREATE, objects=[{type=APPOINTMENT, objConds=[attendees="Paul Compton", time="11:00 10/07/06", title="Project archive"]}]]
Respond	11	Calendar - SPA	domain=CALENDAR, objects=[{type=APPOINTMENT, id=123, objConds=[attendees="Paul Compton", time="11:00 10/07/06", title="Project archive"]}]]
Respond	12	SPA - User	domain=CALENDAR, objects=[{type=APPOINTMENT, id=123, objConds=[attendees="Paul Compton", time="11:00 10/07/06", title="Project archive"]}]]

Table 4.12: Sample Discourse Segment's Stack Entries

4.3.5 Response Generation

The system's responses can be in different modalities: speech, graphical contents, text, etc. For each domain, there is a set of *Response Generation* plans, which are capable of generating the results in different formats. For example, there are two plans for returning responses in the e-mail domain: one returns the contents

of an e-mail message if the task result contains only one message (Figure 4.24), the other, on the other hand, returns a list of message headers if the task result contains more than one message. The same data structure (Table 4.11) is used for representing GUI input and output. Example GUI output generated by these two plans is shown in Figure 4.26.

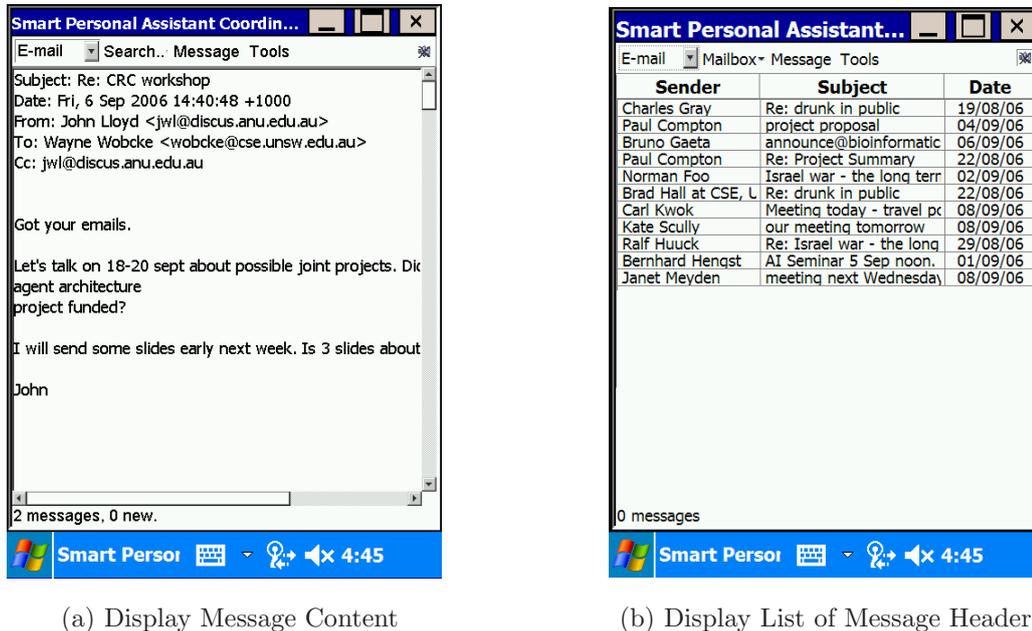
```
Handle Event:
    GenerateEmailResponseEvent
Relevance:
    TaskDomain is EMAIL
Context:
    TaskResult contains 1 message and
    UserDevice is Desktop
Body:
    return MessageContent to UserDevice
```

Figure 4.24: Return Message Content JACK Plan

```
Handle Event:
    GenerateEmailResponseEvent
Relevance:
    TaskDomain is EMAIL
Context:
    TaskResult contains 1 message and
    UserDevice is PDA and
    MessageLength is long
Body:
    generate MessageSummary
    return MessageSummary to UserDevice
Fail:
    return MessageContent to UserDevice
```

Figure 4.25: Return Message Summary JACK Plan

Because the SPA allows users to interact through a variety of devices, there are two plans for displaying e-mail contents, one for use in the context that the user is on the Desktop and another for when the user is on a PDA, which summarizes the message first before displaying it. The context of the plan is



(a) Display Message Content

(b) Display List of Message Headers

Figure 4.26: GUI Output for E-Mail Displaying Task

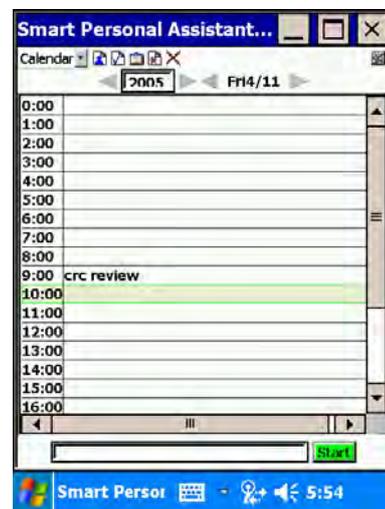
important for deciding which plan to execute. Figure 4.25 illustrates a dialogue plan for generating and returning the summary content of an e-mail message. The context which makes this plan applicable is that the user is on a PDA and the length of the message is too long to be displayed on one PDA screen. Information about the user's current device can be obtained from the *user model*. The plan is to summarize the message content to one screenful in size and then return the summary to the device. The Dialogue Manager could learn to adapt its responses according to the user's preferences such as displaying a summary if the e-mail message is long and not important. The definition of "long" could also be learned. This problem of learning to generate adaptive responses motivates the integration of learning into the Dialogue Manager agent, which will be discussed in Chapter 5.

The speech responses are generated according to the task domain, the type of the requested task and the characteristics of the objects in the result. The Dialogue Manager uses domain-specific knowledge from its beliefs and pre-defined linguistic templates for language generation. The use of domain-specific knowl-

User Do I have appointment with Kate tomorrow.
 SPA You have an appointment with Kate tomorrow about workshop.
Displays calendar view for tomorrow, Figure 4.27a.
 User Cancel that appointment please.
 SPA Are you sure you want to delete that appointment?
 User Yes.
 SPA Appointment has been deleted.
Displays the same calendar view, tomorrow, Figure 4.27b.



(a)



(b)

Figure 4.27: GUI Output for Appointment Cancellation Task

edge is necessary in generating system responses, in particular, generating graphical contents. For instance, Figures 4.27 and 4.28 illustrate examples of two similar tasks in the calendar and e-mail management domains: cancelling appointments and deleting e-mail messages. For the calendar domain, after the appointment has been cancelled, the system just needs to re-display the same day and the correct view will be updated automatically. In the case of the e-mail domain, the Dialogue Manager must use the *discourse history* to determine what messages to show.

User Show me my e-mail from John.
 SPA You have two messages from John Lloyd.
Displays two message headers, Figure 4.28a.
 User Delete the first one please.
 SPA Are you sure you want to delete that message?
 User Yes.
 SPA Message has been deleted.
Displays the other message header, Figure 4.28b.



(a)



(b)

Figure 4.28: GUI Output for E-Mail Deletion Task

4.4 Concluding Remarks

Before closing this chapter, we would like to re-emphasize that in our agent-based approach to dialogue management, the dialogue model is based on speech act theory and consists of a set of modular plans that specify the Dialogue Manager's actions for handling different conversational acts or performing domain tasks. The modularity of the dialogue model, therefore, enables the reuse of discourse-level plans in other applications. In addition, it facilitates the addition of new back-end task assistants into the system because this requires only the development of some domain-level but not generic discourse-level plans. This

has been evidenced by the fact that we have successfully extended without much effort the first version of the Dialogue Manager (with only one e-mail task assistant) for handling an additional calendar management domain. Importantly, the control flow of the dialogue is derived automatically in a flexible way during the BDI interpretation cycle using the agent's plan selection mechanism. Hence it is possible to integrate learning into the agent architecture for learning plan selection mechanism, thus achieving adaptivity in dialogue interaction and processing.

AN ADAPTIVE AGENT-BASED DIALOGUE MANAGER

In this chapter, we consider the problem of dialogue adaptation in the Smart Personal Assistant (SPA) system, more specifically enabling the system to tailor its responses according to the conversational context, the user's physical context and the user's devices and preferences. We have used a BDI agent architecture for the development of the SPA's Dialogue Manager, the central component that maintains coherent dialogue with the user as well as coordinates the back-end task assistants. The Dialogue Manager has a set of modular plans, each for handling a particular discourse-level or domain-level goal. In addition, the information about the conversational context and other knowledge is maintained in the agent's internal beliefs. In this chapter, we show that achieving dialogue adaptation in the SPA can be considered the problem of learning plan selection strategies. We present in this chapter a general method for incorporating learning into a BDI agent architecture so that the agent can learn to select the most suitable plan among those applicable, taking into account the current context and other conditions. An example will be given, illustrating how dialogue adaptation could be used in the SPA. Because the technique is general, it can be applied to other similar learning problems.

Firstly, in Section 5.1, we discuss our motivation for considering the dialogue adaptation problem. As our approach to dialogue management is based on using a BDI agent architecture, we look at the adaptation problem from two aspects: achieving dialogue adaptation and learning in BDI agents. Existing work in

these two areas will be discussed. Next, we present our method for incorporating learning into the agent’s plan selection decision-making process. By learning plan selection strategies, the agent is able to select the most appropriate plan in the current context.

Secondly, in Section 5.2, we describe our work in applying this method to produce adaptive dialogue responses in the case of the SPA. We have integrated the Alkemy symbolic learner [Lloyd, 2003] into the SPA’s Dialogue Manager agent to develop an adaptive dialogue agent, which is able to tailor the system responses according to the conversational context and the user’s device and preferences.

5.1 Dialogue Adaptation

Adaptation in dialogue systems is needed to provide appropriate user-system interaction, resulting in more user satisfaction. With the use of dialogue systems on mobile devices, the need for adaptivity is even more important. The system’s responses must be adapted to the conversational context, the physical context of the user, and the limitations of the device itself, taking into account the individual user’s needs.

In different conversational contexts, the same information may need to be presented to the user in different ways. For example, suppose the user has asked the system to look for *“new e-mail messages from John”*. There are messages from two Johns: John Lloyd and John McAfee, and the system must display the headers of those messages on the user’s device. Thus the message headers should be sorted by sender so that messages from different Johns can be easily seen. In fact, it happens that both Johns are involved in the same group project, CRC. If the user’s request was to get *“all new messages from CRC group members”* then the message headers might instead be sorted by priority or arrival time.

Users of desktop PCs must stay at home or in the office in order to use their computers; they are now able to access information from any location at

any time using mobile devices. These devices can be used in different physical environments. Some places are quiet enough for interaction through spoken dialogue, while it would be impossible in other noisy environments such as train stations or shopping centres. In some cases, spoken dialogue might not be suitable because of privacy problems and/or social inappropriateness, e.g. during a seminar. Therefore dialogue systems should be aware of the user's physical location (perhaps making use of their calendar) in order to adapt its interaction accordingly.

The effectiveness of user interaction using a device with small screen and limited bandwidth can be improved by reducing the amount of information returned to the user [Billsus *et al.*, 2002]. E-mail messages are often too long to fit on one PDA screen. Thus showing only a message summary could sometimes be more desirable. Similarly, in different conversational contexts, the same information might be presented to the user in different ways, e.g. using a different modality, format or layout. For instance, a search result may contain so many messages that all the headers can not be displayed on one PDA screen. The system may offer the user to show only the first few of the results or the messages of high priority.

In addition, dialogue adaptation can allow more appropriate dialogue strategies or dialogue actions. In some cases, it may be necessary that the system should ask the user for confirmation before performing tasks, e.g. deleting e-mail folders, to avoid mistakes. However, whether a confirmation is needed depends on the user's preferences. Another question is whether and when the system should interrupt the user to provide notifications. The system may choose to immediately notify the user or defer the notification because prompt notifications might distract the user from other ongoing tasks. Again the user's preferences should be considered in making the appropriate decision.

In the case of the SPA, in which mobile devices such as PDAs are of interest to address the issue of mobility, we focus on the particular problem of content adaptation to overcome the limitations of the devices, thus enhancing the sys-

tem's usability. Because the SPA's Dialogue Manager uses plans for generating responses, the problem of dialogue adaptation becomes the problem of learning which dialogue plan should be selected and executed in a given context. The input parameters to be considered in this learning problem include the conversational context and the user's device, physical context and preferences.

5.1.1 Existing Work in Spoken Dialogue Adaptation

There has been comparatively little work done in the area of adaptation in spoken dialogue systems. Among the existing work, the forms of adaptation belong to either one of two categories: the problem of adapting the system's questions for eliciting user preferences and the problem of learning strategies in a game-theoretic approach to dialogue modelling.

Recent work in the first category is the ADAPTIVE PLACE ADVISOR by Thompson *et al.* [2004], a recommender system in which the system performs database queries for domain-specific information and asks questions to eliminate domain objects from consideration. User preferences about object attributes and attribute values are learned from the constraints that are mentioned in the user's responses. Questions to the user will then be adapted to both short and long-term preferences, and hopefully can reduce the number of interaction steps in each dialogue. This form of adaptation is domain-specific, more towards improving the system's problem-solving performance than improving the dialogue interaction.

An example of dialogue strategy learning is the Email Voice Interactive System (ELVIS) by Walker *et al.* [2000], which follows a simple state-machine approach for dialogue management in which each node corresponds to a dialogue state. For each strategy type (e.g. dialogue initiative, summarization task, etc.) there are a number of possible strategy choices leading to different state transitions. Reinforcement learning was used to learn the strategy choice selection using a training corpus of collected dialogues. In extending the work of Walker *et al.*, Frampton and Lemon [2006] aimed to explore the use of reinforcement

learning in learning dialogue strategies for frame-based dialogue systems. In their work, slot-filling update rules are defined based on a number of dialogue moves so that the dialogue moves and the status of the information slots can be used as the learning features. The dialogue corpus obtained from previous work was annotated to learn user simulations. The simulations are then used for both training (50000 dialogues) and testing (1000 dialogues) the learning problem, which shows a significant improvement of 65.9% using a scoring function based on task completion and dialogue length rewards.

Similarly, Jokinen and others in the Interact project [2002] also employed reinforcement learning to learn dialogue strategies. Interact's dialogue model is more complex than that of the ELVIS but less complex than that of the SPA. It consists of a set of agents, each corresponding to a system action or dialogue act. The system is trained to learn a table of real values which is indexed by dialogue states. In interacting with the user, the dialogue agent with highest value is selected for the current dialogue action.

Both the Interact and ELVIS systems rely on training dialogues to build its adaptivity model. In case of the SPA, in addition to the conversational context, dialogue adaptation must also take into account the user's preferences as well as the system's physical context. Thus it can not be trained using a pre-collected training corpus. The dialogue model of the SPA is more complex, with different types of dialogue plans, and the control flow comes automatically as the result of the agent's plan selection. Therefore, the SPA requires an adaptivity model that can acquire the plan selection strategies of the Dialogue Manager agent.

5.1.2 Existing Work in BDI Agent Learning

There have only been few attempts to incorporate learning within BDI models of agency, of which none has looked at the problem of how the agent learns to select the most appropriate plan among those applicable in a certain context. All of the existing work provides a loose coupling of the agent architecture and

the learner while for the problem of learning plan selection a close integration is required.

Guerra-Hernández *et al.* [2004] propose an extended BDI agent architecture that provides a mechanism for learning from past experience if a plan succeeds given the current context. Both the agent's beliefs and the plan context are expressed in a first-order language. An inductive logic decision tree learner is used to learn how the beliefs of an agent (i.e. the effective context of a plan when instantiated) may indicate the plan execution's success or failure. In a multi-agent system of such agents, the agents can request or exchange training examples. The system developer uses the learning result to modify the plan definition accordingly. This work is perhaps the closest to ours. However, our research problem is different as it requires the agent be able to learn why a plan should be *selected* in a certain context. Moreover, the agent must be able to learn and adapt in real-time without intervention from the agent developer.

Phung *et al.* [2005] present a learning model within a BDI framework, developed independently of our work, which allows the agent to choose the most suitable strategy for achieving its goal in a simulated robotics domain. An example strategy could be which retardant should be used in a fire extinguishing scenario. In this work, a learner is attached to the agent that learns new *virtual beliefs* from the agent's past experience. The agent uses this learned knowledge in selecting its plans. Our work, on the other hand, aims to achieve adaptive plan selection using meta-reasoning.

Other research proposed the integration of case-based reasoning (CBR) with a BDI architecture [Olivia *et al.*, 1999; Corchado *et al.*, 2004], in which the BDI decision-making process is implemented using case-based reasoning. This approach is not suitable for our research problem because we do not intend to change the BDI practical reasoning behaviour of the SPA's Dialogue Manager agent. Similar to the work by Phung *et al.*, this approach is only applicable for isolated learning problems at the level of an individual agent acting without dynamic changes in the environment.

5.1.3 Adaptation as Learning Plan Selection

In our agent-based approach to dialogue management (Chapters 3 and 4), the conversation flow is controlled using the agent's plan selection mechanism. That is, to achieve a goal, the Dialogue Manager selects and executes appropriate dialogue plans according to the conversational context. Thus to build an adaptive dialogue manager, there are two problems to be solved: (i) defining the possible system actions in any given situation, and (ii) defining the mechanism for the system to decide when to choose which action. Using our agent-based approach, system modularity is facilitated through the use of plans, each handling some dialogue aspect or domain task. Hence the possible choices of system action can be modelled as different plans applicable in the same situation. In addition, a learner can be integrated into the agent architecture and used for the selection mechanism to choose the most suitable plan according to the agent's beliefs, which represent the user's context and/or preferences. Thus the learner accepts as input a set of possible plans and some of the agent's beliefs and outputs the most suitable plan(s). Figure 5.1 gives a high-level picture of how learning can be incorporated into the agent's plan selection decision-making process in the PRS architecture (Section 3.2.1).

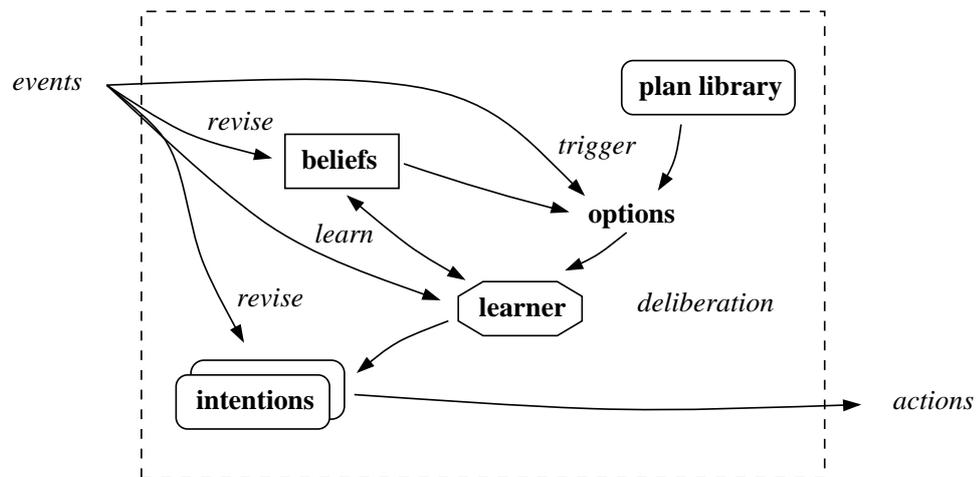


Figure 5.1: Integrating Learning into the PRS Agent Architecture

A learner is integrated into the architecture and used in an intermediate step of the deliberation process. The knowledge of the learner is considered as a part of the agent's internal beliefs. External events from the environment and the agent's internal events trigger the selection of applicable plans from the plan library. During the deliberation process, the agent must select one from these plans for later execution. In order to select the most suitable plan, the set of applicable plans and some of the agent's beliefs such as the attributes of the environment are used as the input to the learner. There are two cases as follows:

- If no result can be found, an appropriate event is raised which triggers the execution of a special plan. In executing this plan, the agent's actions affect the environment so that feedback can be obtained in form of new external events. As a result, the learner revises the learning model.
- Otherwise, the resulting plan, i.e. the most appropriate choice according to the learner's knowledge, is adopted and added to the agent's intention structure.

Thus incorporating learning into the agent architecture can allow the agent to learn to select the most appropriate plan in the given context.

5.2 An Adaptive Dialogue Agent

In this section, we describe our work in extending the SPA's Dialogue Manager by integrating learning mechanism into the agent, allowing the Dialogue Manager to tailor the system responses according to the conversational context and the user's device and preferences [Nguyen and Wobcke, 2006a]. Not all learning methods are suitable for dialogue management. First, learning adaptation strategies in dialogue systems should be conducted in an unobtrusive manner without impacting the user's ordinary use of the system. In addition,

as the training data is limited, user-dependent and cannot be obtained in advance, learning methods that are statistics-based or reinforcement learning are impractical. We have chosen Alkemy [Lloyd, 2003], a symbolic inductive learner, for the implementation. Alkemy offers an expressive way of representing background knowledge using higher-order logic. Moreover, Alkemy also supports online incremental learning. In comparison to other well-known symbolic machine learning techniques such as ID3 [Quinlan, 1986], FOIL [Quinlan, 1990] and Progol [Muggleton, 1995], Alkemy allows the predicate search space (hypothesis space) to be defined and enumerated in a clear way using a predicate rewrite system. A brief introduction to the Alkemy learner will be given in Section 5.2.1.

The Dialogue Manager of the SPA is implemented using a Java-based PRS-style agent platform, JACK Intelligent AgentsTM [Howden *et al.*, 2001]. JACK provides meta-level reasoning as a way to overwrite the agent's plan selection mechanism, making the integration with learning easier. In JACK, goals are special internal events that the agent posts to itself that trigger plan selection and execution. If there is more than one applicable plan, JACK posts a special *PlanChoice* event which must be handled by a special *meta-plan*. The learner can be queried during this meta-level reasoning process to select the appropriate plan according to the response adaptation model that has been learned. Feedback obtained from plan execution failures or from the user can be used to update the learner, i.e. update the response adaptation model.

5.2.1 The Alkemy Learner

Alkemy is a inductive symbolic decision-tree learner which differs from other conventional decision-tree learners in two respects. First, the learning individuals are represented in a typed, higher-order logic, not feature vectors. This supports representation of domain data with complex structure. Secondly, Alkemy provides a highly-expressive predicate rewrite system to constrain the predicate space. Alkemy constructs a decision tree as follows. To split a tree node with

the current corresponding set of examples S , the predicate space is searched efficiently to find a suitable predicate p that induces the best binary partition of S . Starting from this newly-created node, the two sub-trees will then be constructed similarly using the corresponding partitioned subsets of S . We are using the on-line version of Alkemy, in which training examples are given to the learner over time. However, Alkemy considers only recent training examples contained in a fixed-size window for retraining the decision tree. At any time Alkemy can be asked to predict the class of some unseen individual based on its current decision tree. Thus Alkemy is able to build the decision trees incrementally. This is an important feature because in the case of the SPA, it is impossible to obtain training dialogues in advance so that the dialogue adaptation model must be updated on-line and incrementally. Moreover, it is possible for the learner to “unlearn” incorrect generalizations by providing a suitable window size so that only “recent” examples are contained within the training window. We use a window size of 100 training examples. Furthermore, it is also worth mentioning that in Alkemy, numeric values cannot be learned but have to be pre-defined in the problem specification given to the learner.

5.2.2 Adaptive Dialogue Responses

In order to show the applicability and feasibility of the proposed approach, we aim at adapting the dialogue responses to the conversational context and the user’s devices and preferences. We focus the learning problem on the e-mail management task domain. The Dialogue Manager uses a set of plans (the *Response Generation* plan group) for generating responses in different formats: a message’s content or summary; a subset of message headers so that the list fits on the PDA screen; or a full list of message headers sorted by sender, priority or folder. In addition, knowledge of the user’s current device is kept in the agent’s beliefs and updated when needed. Given a task result, if there are a number of applicable plans for generating the system responses, the user is

asked to choose one among them. The chosen plan is selected and executed, while the user's preferences are obtained and input to the learner for updating the decision tree. Hence, gradually over time, the system will be able to learn the user's preferences and adapt the dialogue responses appropriately. It is also possible that the user's preferences can be obtained by observing the user-system interaction and considering how the current user request fits with the current context, i.e. how the current request relates to the previous completed task. However this has not been considered in this work.

Figure 5.2 shows the interaction of the Dialogue Manager's *Response Generation* plan group with the Alkemy learner. For example, the Dialogue Manager may display the whole content of an individual e-mail message on the device by executing the *Return Message Content* plan. However, it is possible that the message is summarized first then presented to the user by using the *Return Message Summary* plan. If the result is a list of messages, the dialogue manager can display the message headers on the device and sort them by arrival time (default), sender, folder or priority, using corresponding plans. With dialogue adaptation enabled, the *Response Generation Meta Plan* is triggered whenever the system needs to generate a response. Within the meta-plan, learning individuals are generated and sent to the Alkemy learner for prediction. Alkemy predicts for each available *Response Generation* plan whether it is a possible choice or not. If there is more than one possible choice then the *Request User Clarify Preference* plan is triggered asking the user for a selection. The user's response is then processed by the *Preference Processing* plan and Alkemy is called to update the decision tree. In other words, the Dialogue Manager agent specifically asks the user to select one preferred action from the set of applicable plans. However, once the selection has been made, the agent can still unlearn that selection. If in a similar situation, the user requests the system to select a different plan, the agent would be able to learn this new selection based on the new positive example and the negative example associated with the previous selection.

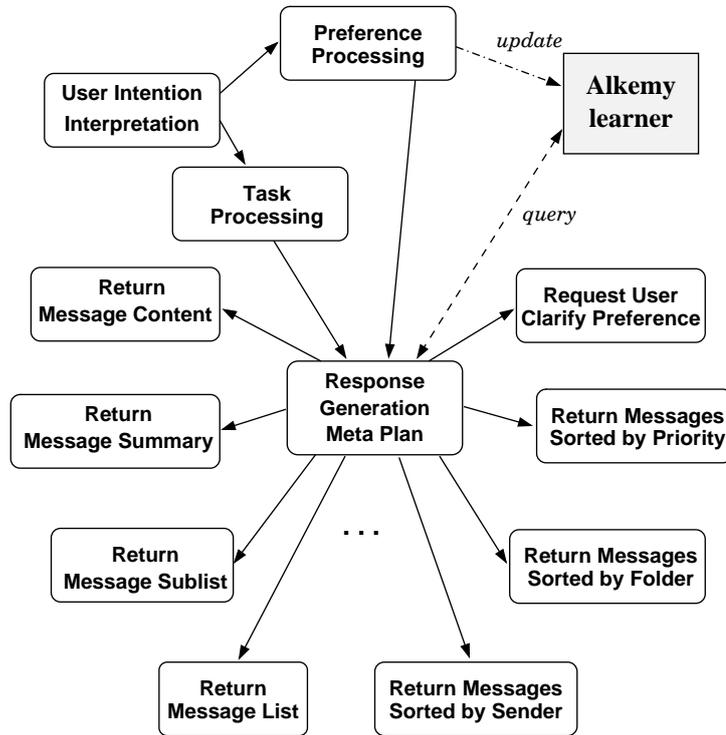


Figure 5.2: Learning Plan Selection for Response Generation

Decision tree learners are not able to learn a set of values. However, in our learning problem, it is necessary to predict more than one possible plan. Therefore, given the current context, for each available *Response Generation* plan, a learning individual is generated which includes information of the current context as well as the plan name. The individuals must be classified to be either *True* or *False*. If the classification result is *True*, this plan is considered an option. Otherwise, the plan does not meet the user's preferences in this particular context.

If there is more than one plan classified *True*, they are all considered options according to the user's preferences. The *Request User Clarify Preference* plan is executed, asking the user to specify the preferred system action, i.e. the most preferred plan. For example, the clarification request might be “*Do you want to sort the result by sender or folder?*”. The user's response is then interpreted and processed by the *Preference Processing* plan to update the learner accordingly. If there is only one plan classified *True*, the plan is immediately selected and

executed for returning the response to the user. However, the selected plan may not be the user's preferred option. For instance, suppose the system has decided to show the summary of an e-mail message and the user requests “*No, show me the whole message.*”. New training examples will be sent to the learner for updating the decision tree, and the previous examples which are no longer valid will be gradually removed from the training window.

The Alkemy specification of the learning problem includes the data constructors as given in Figure 5.3. The *Mode* element indicates the current modality of the dialogue, where *Text* corresponds to typed text and *None* means the user is currently using only the GUI. The *Task* element is used to represent the current domain task and the *Device* element, the user's current device. Similarly, plans in the *Response Generation* group are defined using the *Plan* element. Other elements represent different features of an e-mail message such as the sender, the length and the priority, etc.

Each learning individual (training example) contains information of the current context (e.g. the user's current device, the modality of interaction, the domain task, etc.), the task results (i.e. the set of e-mail messages) and the name of the applicable *Response Generation* plan to be considered. This is pre-computed and sent as input to the learner. The learning individual is of the *Product* type and is defined as follows:

$$\begin{aligned} Individual &= Device \times Task \times Mode \times TaskResult \times Plan; \\ TaskResult &= Set\ Email; \end{aligned}$$

In which, the task result is a set of e-mail messages. Each message is represented by four components: the sender name, the number of lines in the message's content, its classified virtual folder and the priority:

$$Email = Sender \times Length \times Folder \times Priority;$$

The learning problem is to predict whether the plan is an option in the current context according to the user's preferences. If the classification result is *True*, in this situation the plan meets the user's preferences (which have been learned)

The transformation below returns true (of type *Bool*) if there exists any message in an e-mail set (of type *Set Email*) that satisfies some condition (specified as $Email \rightarrow Bool$):

$$\begin{aligned} & setEmailExists : (Email \rightarrow Bool) \rightarrow (Set Email) \rightarrow Bool; \\ & setEmailExists : setexists(1); \end{aligned}$$

The set of e-mail messages is defined as the fourth element of a learning individual. The transformation *projEmails* below extracts this e-mail set from the learning individual:

$$\begin{aligned} & projEmails : Individual \rightarrow SetEmail; \\ & projEmails : project(3); \end{aligned}$$

Suppose the transformation *lt30* returns true if a number is less than 30. Combining *projEmails*, *setEmailExists*, and *projLength* transformations gives the following predicate which returns true iff there exists at least one message in the e-mail set whose length is less than thirty lines:

$$projEmails \circ setEmailExists(projLength \circ lt30)$$

Applying the *numOfEmails* transformation to an e-mail set returns the number of messages in that set: *projEmails* \circ *numOfEmails* (*true*). The message length is only useful if there is exactly one message in the result. The following predicate is true iff the e-mail set contains exactly one message whose length is less than thirty lines:

$$\begin{aligned} & and (projEmails \circ numOfEmails (true) \circ eq1) \\ & (projEmails \circ setEmailExists (projLength \circ lt30)) \end{aligned}$$

The number of distinct senders in the result e-mail set is also useful information which can be obtained from the following transformation, in which the *setDomCard(true)* transformation returns the set's cardinality.

$$projEmails \circ projSetSender \circ setDomCard (true)$$

Expanding (5) or (6) results in predicates for checking the length or the priority of the only e-mail message in the result. Similarly, (7) is used for constraining the total number of messages while (8) is for the number of distinct folders. The predicate space is efficiently searched by Alkemy to construct the decision tree in which each node corresponds to a predicate that best partitions the training examples at that point. Note that defining this predicate rewrite system requires the expression of domain-specific knowledge. Furthermore, constraining the learner's hypothesis space is necessary because of the limited availability of training data. In Section 1 of Appendix B, we give the full specification of the learning problem used for Alkemy. Example training data collected is shown in Section 2 of Appendix B. More detail on how Alkemy constructs the hypothesis space and the decision tree can be found in [Lloyd, 2003].

User Is there any new mail from Wayne?
SPA You have one new message from Wayne Wobcke.
SPA *Displays the message from Wayne Wobcke.*
User Find all messages about meeting in the Inbox.
SPA There are twenty two messages about meeting in your Inbox.
 Displays the headers of the twenty two messages.
User Show me the one from John.
SPA Here you are.
 Shows the content of the message from John Lloyd.

Figure 5.5: An Example Dialogue without Learning

Figure 5.5 shows an example dialogue without adaptation. Assuming that the user is using a PDA for interacting with the SPA, with dialogue adaptation enabled, this example dialogue would be as given in Figure 5.6. Note from this dialogue how the Dialogue Manager has learned to adapt its responses according to the user's preferences. The Dialogue Manager has in its plan library two different plans for returning a response containing one e-mail message to the user: *Return Message Content* for generating the whole message content and

User Is there any new mail from Wayne?
SPA You have one new message from Wayne Wobcke. The message is more than thirty lines, should I just show you the summary?
User Yes please.
SPA *Displays the summary of message from Wayne Wobcke.*

(a)

User Find all messages about meeting in the Inbox.
SPA There are twenty messages about meeting in Inbox folder. I'm displaying the first ten messages.
Displays the first ten message headers.

(b)

User Show me the one from John.
SPA Here is the summary of the message from John Lloyd.
Displays the summary of message from John Lloyd.

(c)

User Show me the whole content.
SPA Here you are.
Displays the content of message from John Lloyd.

(d)

Figure 5.6: Dialogue with Learning

Return Message Summary for displaying only the message's summary. In the given scenario in example (a), both these plans are applicable because there is only one new message from Wayne. For each plan, a query is sent to the learner to check if the user would prefer the system to use that plan. For example, the query for the *Return Message Summary* plan is as follows, in which 60 is the length and NORMAL the priority of the message.

```
? ResponsePlan (PDA, SEARCH, SPEECH,  
                {"Wayne Wobcke", 60, "Inbox", NORMAL}),  
  ReturnMessageSummary) ;
```

The learner returns *True* for both queries because the system does not know the user's preference in this case. The system randomly chooses to use the *Return Message Summary* plan, suggesting to show only the message's summary. The user agrees with the system's suggestion, indicating his/her preference. The two training examples shown below, are then formulated and sent to the learner.

```
ResponsePlan (PDA, SEARCH, SPEECH,
              {"Wayne Wobcke", 60, "Inbox", NORMAL}),
              ReturnMessageSummary) = True ;
```

```
ResponsePlan (PDA, SEARCH, SPEECH,
              {"Wayne Wobcke", 60, "Inbox", NORMAL}),
              ReturnMessageContent) = False ;
```

These individual learning instances are added into the training set and used for updating the decision tree. In this case, the *lt30* predicate can be used as a hypothesis to distinguish these examples. As a result, the system has learned to show the summary if the message body is more than thirty lines and a typical example of a decision tree representing the user's preferences which results in such adaptive behaviour is given in Figure 5.7.

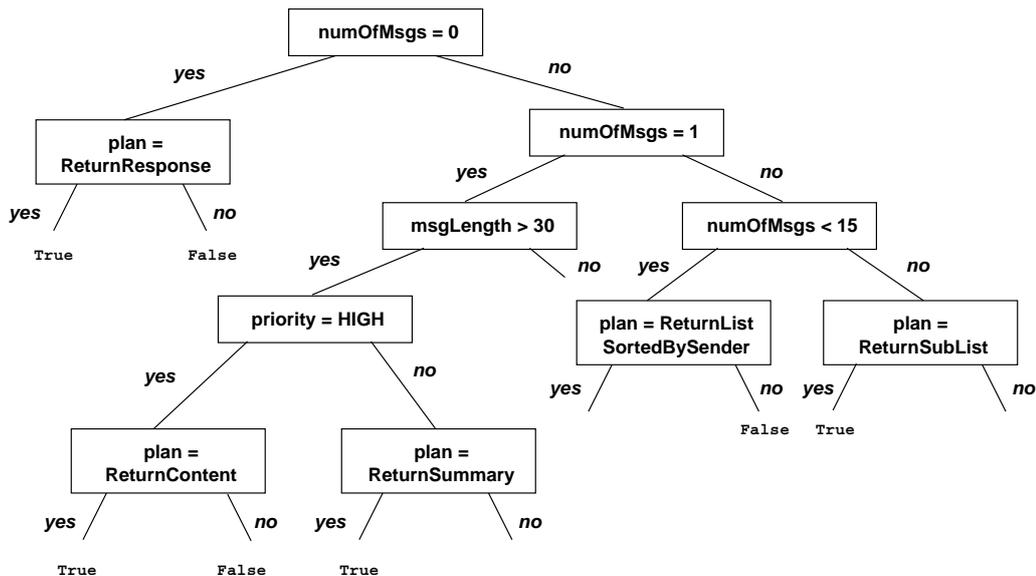


Figure 5.7: Decision Tree Representing User's Preferences

The decision tree can be understood as follows: if there is no message in the result then use the simple *Return Response* plan to generate the response utterance without showing anything on the GUI; if there is exactly one message and its length is greater than thirty lines then use the *Return Summary Content* plan; however, if the message has high priority then do not use the summary; if the total number of messages in the result is less than fifteen, sort them by sender; otherwise, if there are more than fifteen messages, use the *Return Sub List* plan to show just the first few message headers.

It is also important that the system allows the user to correct its actions. In responding to a show e-mail task, the system is showing the user a message summary (according to the learned user preferences), but if the user explicitly asks to see the whole content, the system is able to adjust its response and update the learning model. Later on in the dialogue (example (c)), the system immediately shows the summary of the e-mail message from John Lloyd because its length is also more than thirty lines. However, because this message is of high priority the user asks for the whole message content, so the system displays the content and learns a new preference (example (d)). In this case, the following training examples are generated by the Dialogue Manager:

```
ResponsePlan (PDA, SHOW, SPEECH,  
              {"John Lloyd", 77, "Inbox", HIGH}),  
              ReturnMessageSummary) = False ;  
ResponsePlan (PDA, SHOW, SPEECH,  
              {"John Lloyd", 77, "Inbox", HIGH}),  
              ReturnMessageContent) = True ;
```

Similarly, assume the SPA has learned in the past, using a similar mechanism, to show only the first 10 message headers if there are 15 or more messages in the result (resulted in the decision tree in Figure 5.7). Thus in example (b), only the first 10 message headers are displayed to the user.

5.3 Summary

In this chapter, we have discussed the problem of achieving adaptive response generation in dialogue systems, which can be recast as the problem of learning plan selection strategies, using our agent-based approach to dialogue management. We have presented a general method for incorporating learning into a BDI agent architecture, and described our work in applying this method to the case of the SPA's Dialogue Manager. The Alkemy learner has been integrated into the Dialogue Manager to achieve an adaptive dialogue agent, which is able to tailor its responses according to the conversational context and the user's device and preferences.

In dialogue applications such as the SPA, generating training data for learning adaptive responses is an important issue. Dialogue adaptation in the SPA must take into account the user's preferences as well as the system's physical context, thus cannot be trained using pre-collected training corpora. Currently, the user is asked explicitly for the preference if there are a number of possible actions. However, future work could be to obtain the training data by observing the user's interaction. The relation between the user's current request and the current context could suggest the user's preferences. For example, suppose the user has just asked for new e-mail from John Lloyd and there is only one message that is of high priority. Next, the user immediately requests to read out the message content. This may indicate that if there is one important message from John Lloyd, the user prefers to see the message's content as well as hear the message read out, or the user does not want to read the message because the PDA screen is too small. Another problem is to unlearn incorrect generalizations. The current implementation uses an on-line version of Alkemy that allows the decision tree to be learned from a fixed-size set of recent training examples. As old training examples are gradually discarded from the training set, it might be possible to estimate a suitable window size to minimize generalization errors.

EVALUATION

In this chapter, we discuss the evaluation of the Smart Personal Assistant (SPA) application, focusing on the evaluation of the dialogue model of the SPA. In addition, we are interested in understanding how the performance of various components including the speech recognizer, the Partial Parser, the back-end applications, and importantly the Dialogue Manager, contribute to the overall performance of the system. Firstly, in Section 6.1, we present existing frameworks for evaluating spoken dialogue systems. Next, we describe in Section 6.2 how the evaluation of the SPA was set up, concerning the choice of the subjects and the evaluation methods and metrics. The experimental results and explanation are reported in Section 6.3. Finally, our additional observations and comments are presented in Section 6.4.

6.1 Evaluation of Spoken Dialogue Systems

According to Bernsen and Dybkjær [1998, p. 191], there are three types of evaluation of spoken dialogue systems: *performance evaluation* for measuring the performance of the system in terms of some quantitative parameters, *diagnostic evaluation* to detect design and implementation errors, and *adequacy evaluation* for evaluating how the system meets user needs and expectations. Perhaps for research prototypes, the performance and adequacy evaluations are more often required in order to justify the design of the dialogue model and to evaluate the usability of the whole system. Evaluation can be *subjective*, based on the users' opinions, or *objective*, to produce subject-independent assessment [Dybkjær and Bernsen, 2001]. In brief, evaluating spoken dialogue sys-

tems requires consideration about the types and purposes of the evaluation, the methods, the subjects, and finally, what and how to evaluate. However, there is currently a lack of a general evaluation framework, making it difficult to evaluate and compare dialogue systems. We now consider two existing approaches and whether they can be adapted to suit our evaluation purposes: the PARADISE framework of Walker *et al.* [1997] and the three-tiered model of Stibler and Denny [2001].

PARADISE uses a weighted function to combine the measures of individual performance objectives, such as user satisfaction, task success, dialogue cost, etc., into a single performance evaluation function for a whole system. The evaluations corresponding to these performance objectives are done as follows:

- *Measure of user satisfaction:* The users are asked to fill in a survey. User satisfaction is measured by calculating the mean of user ratings (between 1 and 6), specifying the degree to which the users agree with a statement about the system.
- *Measure of task success:* Each task is represented using an attribute value matrix, representing the information that must be exchanged between the user and the system as a set of ordered pairs of attributes and possible values. For example, in a flight booking domain, the required attributes can be the departure city, the arrival city, the departure time-range (e.g. morning, evening, etc.), and the departure time (e.g. 10 am, 2 pm, etc.). A set of all possible values (for all attributes) is determined and used to build a confusion matrix, counting how many times the system has recognized correctly and incorrectly attribute values in the collected dialogues. The task success performance measure is a function of the frequency of correctly recognized attribute values, taking into account that the chance of a value be misrecognized depends on the size of the value set.
- *Measure of dialogue costs:* Utterances can be classified according to some qualitative criteria such as whether the utterance is a *repair*. Dialogue

cost measures can be the total number of utterances, the number of repair utterances, etc.

Finally, the overall system performance is calculated by combining the task success and other the dialogue cost measures. The user satisfaction is used as the predicted factor for working out the required coefficients using multiple linear regression.

The PARADISE approach is not suitable for complex application domains such as in the case of the SPA. The measure of task success in the PARADISE approach is only suitable for simple slot-filling tasks, e.g. obtaining transport timetables, flight schedules, etc., in which all possible slot values can be predicted. E-mail and calendar management tasks are more complex and cannot be represented using a pre-defined set of attributes and attribute values because the values of attributes such as appointment title, person name, e-mail topic, etc., can be any phrase. In addition, the length of a dialogue or the number of repair utterances does not necessarily reflect the efficiency of the system's dialogue behaviours or the user satisfaction. In evaluating spoken dialogue systems, the contribution of various system components, importantly the speech recognition, must be taken into account. The PARADISE framework does not support this requirement. Finally, it is not even clear whether system performance can and should be meaningfully indicated by a single performance value as the combination of different evaluation measures. Task completion rates/scores and dialogue costs do not necessarily correlate with user satisfaction. In fact, an existing usability study of Frøkjær *et al.* [2000] has suggested very low correlations between the three usability measures: *effectiveness* (e.g. task completion rate), *efficiency* (e.g. task completion time/cost) and *satisfaction* (e.g. user satisfaction). Therefore they should be considered independent aspects in a usability evaluation.

The three-tiered evaluation methodology of Stibler and Denny [2001], in contrast, models evaluation objectives at three levels of abstraction: *user satisfaction* (subjective measures using questionnaire), *system support of task success* (overall system performance based on some definition of "mission success") and

component performance (for assessing an individual component's contribution to the overall system performance). The middle tier requires measurements such as task completion, task complexity, dialogue complexity, task efficiency, dialogue efficiency, task pace, dialogue pace, user frustration and intervention rate. For the lowest tier, evaluation metrics include word recognition accuracy, utterance recognition accuracy, concept recognition accuracy, component speed, processing errors and language errors. We have used an adaptation of the three-tiered methodology for evaluation of the SPA because it provides suitable methods for evaluating the SPA's dialogue model, allowing us to differentiate the contribution of different dialogue management factors in the overall system performance.

6.2 Evaluation of the Smart Personal Assistant

In this section, we describe our evaluation of the SPA, which can be classified into subjective and objective evaluations¹. The methods used were adapted from the three-tiered framework. The main purpose of conducting the subjective evaluation is to answer a general question of whether users would want to use spoken dialogue applications such as the SPA. Although the system has not yet had a commercial deployment, it is robust enough to provide reliable information. The objective results are more important to us, as they can be used to quantify the performance of the Dialogue Manager, thus be used to justify the effectiveness and robustness of the SPA's dialogue management using our agent-based approach.

6.2.1 Evaluation Setup

There were 10 subjects (5 male and 5 female) participating in the evaluation, among them 5 (3 male and 2 female) with an IT background. The subjects were

¹The user study of the SPA were conducted by Anna Wong, who also designed the user questionnaire.

native speakers of Australian English, aged between 18 and 45. The study was divided into training and testing sessions separated by between one and three days. In the training sessions, each subject spent about 30 minutes recording two voice passages (used for constructing a voice model of the user with the Dragon NaturallySpeaking speech recognition software based on the Australian English voice profile). After a short break of 20 minutes, during which the speech recognition profile was built, the subject was given the SPA system and asked to perform 6 simple tasks (Appendix C, Section 1) with some guidance.

The follow-up testing session was carried out after the training and in a lab setting environment so that the user's interaction could be observed through cameras and a one-way mirror. The subjects were asked to re-do the training tasks, and then a series of test tasks (Table 6.1) for us to collect evaluation data. Because the users had never previously interacted with any SPA-like systems, training was essential to allow them to become familiar with the system.

The test consisted of 12 tasks with different levels of difficulty in both the e-mail and calendar management domains, such as searching e-mail messages or scheduling a new appointment. There are also complex tasks that involve more than one of these primitive tasks, for example, finding e-mail messages from a person or about a topic, then deleting these messages. The last task requires context switching between the e-mail and calendar domains. The SPA was configured with the same sets of e-mail messages/folders and pre-defined appointments for all users. However, the e-mail arrival times and appointment times were adjusted to accommodate the task requirements, for example searching for e-mail messages that arrived today (the day of the test session). We intentionally used in the tasks proper names such as "*John Lloyd*", "*Kate*" and "*Jessica*", and phrases for representing e-mail search topics or appointment titles such as "*the war with Israel*", "*conference paper*", etc., to see how well the speech recognizer could handle proper names and these kinds of phrases. For most tasks, correct speech recognition of proper names and phrases is required in order for users to complete the tasks successfully.

6.2 Evaluation of the Smart Personal Assistant

Task	Instruction	Description
1	Find the e-mail from John Lloyd.	<i>E-mail search</i>
2	Find your appointments for next week.	<i>Appointment search</i>
3	Find all e-mails which were sent to you today.	<i>E-mail search</i>
4	View the list of e-mails in the 'Seminars' folder.	<i>Folder search</i>
5	Schedule a meeting with Kathy about the budget, for tomorrow at 11 am.	<i>Appointment schedule</i>
6	Check that you have an appointment on Friday at 11 am. Reschedule it to Monday next week at 2 pm.	<i>Appointment search, reschedule</i>
7	Check that you have an appointment for Tuesday next week at 3 pm. Then delete it.	<i>Appointment search, deletion</i>
8	You are looking for an e-mail from Paul Compton about the project, which you know he sent to you last week sometime. Please find and read his e-mail.	<i>Complex e-mail search</i>
9	Find out what time your appointment with Jessica is today.	<i>Appointment search</i>
10	You have received e-mails about the war with Israel. Please find and then delete all of them.	<i>E-mail search, deletion</i>
11	Find all e-mails about seminars and move them to the 'Seminars' folder.	<i>E-mail search, archive</i>
12	Find your e-mails for today. Read the message from Kate and complete any requests that the sender has asked of you.	<i>E-mail search and appointment search, reschedule</i>

Table 6.1: Evaluation Tasks

The users were instructed to hold the PDA at reading distance and talk into the inbuilt microphone at their normal pace. As we are interested in evaluating the system with speech only, the users were not allowed to use the PDA's stylus except for scrolling and pressing the speech button, before starting and after stopping their utterances. The users were able to see the speech recognition output through a text area on the PDA screen. During the testing sessions, the user's actual speech input was transcribed on the fly and the system processing

logs were saved for objective analysis. The subjective measurements are derived from collected user responses to a questionnaire of 23 questions (Appendix C, Section 2), which the users completed after their testing session.

6.3 Evaluation Results

6.3.1 Objective Results

We consider the objective evaluation at two levels: the task and the utterance levels. The task-level performance measures how well the system performs and fulfils the user requests. The metrics used include dialogue length, task success rate, etc. In addition, we want to know, if a task was not completed successfully, which system component (or components) were the main cause(s) of the failure. At the utterance level, we look more closely at every utterance exchange, i.e. for each user utterance, whether the system response was appropriate, and if it was not, which aspect of the dialogue model was mainly responsible for the problem.

Task-Level Performance

The three-tiered method defines tasks as consisting of segments, each for performing an atomic operation. For instance, in the logistics domain, these operations can be signing in to the system, filling in the order form, submitting the form and signing off. It might be thought that, in the case of the SPA, it is also possible to divide tasks into more fine-grained primitive tasks. For example, the task of *deleting e-mail messages from Paul Compton* can be considered as consisting of a *search* task (for messages from Paul Compton) and a *delete* task (for deleting these messages). However, in many cases, it is not possible to break tasks into segments of primitive tasks because using the SPA, users are able to specify their (though complex) requests in just one utterance. Therefore, our evaluation measures are based on the 12 test tasks given to the users.

Speech recognition accuracy has a great effect on the performance of spoken dialogue systems. We measure the performance of the speech recognizer (Dragon NaturallySpeaking software in dictation mode) by calculating the percentage of *concept-words* correctly recognized. Firstly, let us define that each speaker utterance consists of several *concepts* that are required for the system to be able to fully and correctly recognize the user's intention. In the case of the SPA, these concepts can be the type of the requested task, the name of a folder, the name of the appointment's attendee, etc., or references to task objects. Concept-words are those that denote these concepts, thus they must be present in the speech recognizer's output in order for the system to correctly recognize the concepts, and so the user's intention. For example, the utterance "*Could you show appointments with him for me please.*" contains 4 concept-words: "*show*", "*appointments*", "*with*", "*him*". Note that the SPA's dialogue model is designed to allow flexibility in the use of language, i.e. strictly grammatically correct utterances are not required. Moreover, the Partial Parser is able to recover from some secondary grammatical errors introduced by the speech recognizer. Thus we consider the following two cases as "correctly recognized": plural words misrecognized as their singulars, and verbs misrecognized as of different tenses, both of which occur often. The concept-word recognition performance of the speech recognition system is given in Figure 6.1. The concept-word recognition accuracy ranged between 82.61% for subject (8) and 91.51% for subject (7), with the overall average of 86.94%. Although correct identification of the user's intentions requires 100% concept-word recognition accuracy, the speech recognition results were better than we expected, which allowed the Dialogue Manager to recover from some speech recognition errors, enabling the users to complete most of the tasks.

Figure 6.2 shows the average dialogue length (counting only the number of user utterances taken) of successfully completed tasks. Over all 12 tasks, the users took 4.4 utterances on average to perform one task. Considering the dialogue length for individual tasks, tasks (2), (3), (4) and (9), which are

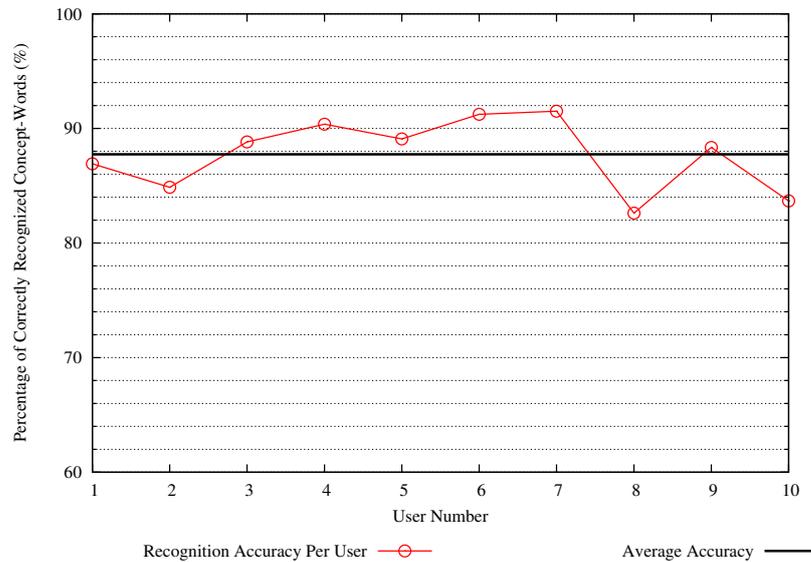


Figure 6.1: Concept-Word Recognition Accuracy

simple e-mail and calendar search tasks, required less interaction, between only 1 and 2 utterances on average. In contrast, tasks (6), (7) and (8), which are compound tasks and more complex than the others, had greater dialogue length, as expected. Similarly, task (12), which involves dialogue interaction in both the e-mail and calendar domains, required an average of 8.25 user utterances. Note that according to the chosen dialogue strategies, *delete* and *reschedule* tasks always involve the SPA requesting the users for confirmation, thus they are more likely to require longer dialogue interaction. It was observed that most subjects tended to prefer breaking the tasks into primitive sub-tasks, though some subjects were able to perform complex tasks in just one or two utterances. Task (10), an e-mail search and deletion task, had an average dialogue length of 7.66 user utterances, which is higher than expected, because of one individual subject who required 23 utterances to complete the task. The reason for this long dialogue was that the speech recognizer continuously misrecognized the word “war” in the e-mail search topic “the war with Israel” as “wall”. The user could avoid using this word, requesting a more general search for messages “about Israel”, but was not aware that this would give the same result.

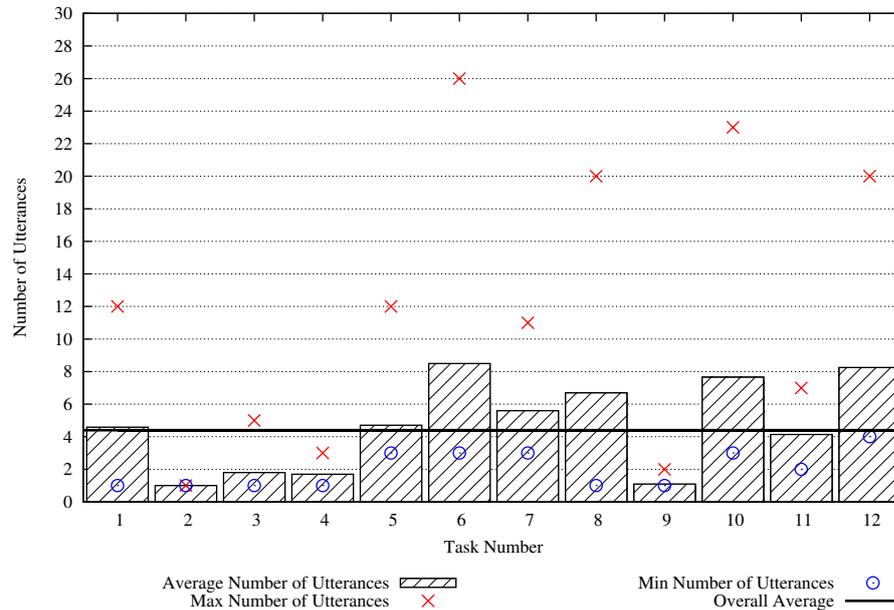


Figure 6.2: Average Dialogue Length Per Task

The maximum and minimum numbers of utterances used for each task are also presented in the graph in Figure 6.2. Taking the example of task (1), a search for e-mail messages from John Lloyd, the worst case was that a subject needed 12 utterances to complete the task because the speech recognizer failed many times to correctly recognize the family name, “Lloyd”. However, there were other subjects, who experienced the same speech recognition error, able to get around this problem by searching for messages from *John* first, e.g. “Do I have any e-mail from John?”. As the result contains only 2 messages, in which only one is from John Lloyd, the subjects could use the message ordering to successfully request the message from John Lloyd, e.g. “Show me the second message”. This shows that some users, perhaps able to gain more experience through the training sessions with the SPA, could interact with the system in more flexible and effective ways than others.

Task (6) is the one with the longest average dialogue length. In this task, the users are required to reschedule a meeting from Friday 11 am to next Monday at 2 pm. Some users did not notice from the beginning that not only the day but the time needs to be changed, thus had to move the appointment twice,

resulting in a longer dialogue length (when changing only the day, the SPA keeps the same time by default). Subject (1) finished the task without realizing that the rescheduled time was wrong. In another case, subject (8) took 26 utterances to complete the task. The speech recognizer produced several crucial errors. Moreover, although the subject had been told in the training session to use the keyword “*about*”, he/she very often referred to appointments using the construction “*the appointment regarding ...*”, which is not currently supported.

Considering the case of task (8), which is a search for a message from Paul Compton that was sent last week, while all subjects completed the task reasonably well, there was one who needed 20 utterances to complete this task. The reason was that the speech recognizer consistently misrecognized the important keyword “*e-mail*”². The subject, user (2), did not think of rephrasing the request using alternative words such as “*message*”. After obtaining the list of all messages from Paul Compton, the subject attempted to request the message sent last week using the exact date sent but, unfortunately, this is not supported by the system. Once the subject realized that he/she could use more natural temporal phrases such as “*last week*”, he/she did complete the task successfully.

In another case, 20 utterances were taken by subject (4) to complete task (12), while the average dialogue length for this task was 8.16. Task (12) is the most complex, requiring context switching between the e-mail and calendar domains. Firstly, it took several utterances before subject (4) could find the message from Kate (sent today) because the proper name could not be recognized correctly. Given that the user was able to view the list of four e-mail messages sent today, he/she could have used the message ordering to request the message from Kate. In the message’s content, the subject was asked to reschedule an appointment with Kate to another time. When requested to confirm the change, the subject had to repeat his/her confirmation, “*yes*”, a few times because of speech recognition errors.

²In his/her accent, “*e-mail*” was pronounced more like “*e-mahl*”.

Table 6.2 presents the actual number of utterances used by each individual user for each successfully completed task. A task is considered not completed (successfully) if either the subject gives up before finishing the task or the system returns incorrect results. For example, subject (5) could not complete task (12), which requires the rescheduling of an appointment with Kate. After many attempts to get the speech recognizer to correctly recognize the proper name “*Kate*”, which is essential to distinguish two appointments with different attendees but the same title and time, the user finally gave up the task. Another example of an uncompleted task was that of subject (2), who thought he/she created the appointment as required in task (5), but in fact, the appointment’s attendee was wrong due to a speech recognition error. We consider this case an uncompleted task because the task result is incorrect.

We also give scores to users according to how well they were able to work out for themselves how to interact with the system. The user scores are presented in the right most column of Table 6.2. If the user performed a task successfully without any guidance, the user is given 1 mark for that task. If some help with wording was required, such as a suggestion on how to rephrase a request, the user receives 0.5 mark for the task (indicated by a star in Table 6.2). The user receives a mark of 0 if the task is not completed successfully (discussed below) or if the user requires too much help (indicated by 2 stars in Table 6.2), such as advice on how to break up a complex task into primitive tasks. The average score for all users was 10.1 out of a possible 12, indicating a very high task completion rate per user.

Over all users and tasks, the task completion rate was 88.3%. There were 14 task failures in the 120 tasks: 3 due to the subject giving up the tasks (indicated by shaded cells), 10 other cases in which the subjects finished the tasks but did not know that the task results were incorrect, and 1 case in which the subject gave up after the system crashed. For these cases, we attribute the failure to the first component that caused the problem. However, when in doubt, the failure is attributed to more than one component. The sources of failure are

6.3 Evaluation Results

Task	1	2	3	4	5	6	7	8	9	10	11	12	Score Per User (out of 12)
	E-mail Search	Appointment Search	E-mail Search	Folder Search	Appointment Schedule	Appointment Search, Reschedule	Appointment Search, Deletion	Complex E-mail Search	Appointment Search	E-mail Search, Deletion	E-mail Search, Archive	E-mail and Calendar Combination	
1	1	1	1	1	3	User	11*	6	1	3	Speech Parser User	Dialogue	8.5
2	6	1	2	3	Speech	7	5	20	1	4	5	5	11
3	6	1	2	1	3	4	5	5	1	Speech Parser	3	4	11
4	2	1	1	3	4	3	3	1	1	4	5	20	12
5	1	1	5	2	4	6	4	1	1	Dialogue	4	Speech	10
6	10**	1	1	1	4	Dialogue	9*	3	1	User	3	5	8.5
7	1	1	1	1	3	10*	4	2	2	Dialogue	Speech Dialogue	8	9.5
8	2	1	1	1	Speech	26*	7	12**	1	23*	7	12	9
9	12*	1	3	2	12	8	3	13	1	9	Speech	5	10.5
10	5	1	1	2	Speech	4	5	4	1	3	2	7	11
Completion Per Task	100%	100%	100%	100%	70%	80%	100%	100%	100%	60%	70%	80%	88.3%
Notation	<i>Shaded cell: task given up by user</i>												
	<i>Text in cell: sources of task failure</i>												
	<i>User score: 1, * 0.5, ** 0</i>												

Table 6.2: Utterances Taken to Complete Tasks, Sources of Task Failure and User Scores

also reported in Table 6.2. Among these cases of task failure, 3 were made by the users, 8 were caused partially or fully by the Partial Parser and speech recognition errors (important words were missed out), and 5 were because of the Dialogue Manager. We now consider these failures in greater detail.

The common problem with three failures of task (5), scheduling a new appointment, was due to speech recognition errors. In two cases, the attendee's name, "*Kathy*", could not be recognized correctly, resulting in incorrect task results. None of the users were given guidance during their training session on how to change appointment titles and attendees. Subject (9), who also had the same problem, however, was able to successfully correct the attendee's name through dialogue using the construction "*Change the attendees*". Subject (10) had a similar problem in that the required title, "*budget*", was misrecognized as "*by jet*". After several attempts using "*Rename meeting*" to change the title, which unfortunately is not recognized by the Dialogue Manager, he/she gave up.

Subject (1) failed to complete task (6) because he/she rescheduled the appointment from 11 am on Friday to the same time on Monday the week after, missing a requirement that the new appointment's time be 2 pm. On the other hand, the failure of user (6) was due to a system crash caused by the Dialogue Manager. Figure 6.3 shows an excerpt of his/her dialogue for task (6). Utterance (5) of the user is intended to contribute to the ongoing sub-dialogue for rescheduling the appointment, requesting to change the appointment's time to Monday at 2 pm. However, because of the verb "*change*", the Dialogue Manager incorrectly recognized the request as to move the appointment (to 2 pm on Monday) as well as to change the appointment's title or attendees. Utterance (7) of the user was not understood by the Dialogue Manager, so the system asked the user again to specify the new title or attendees. Due to an error in semantic analysis, the new title specified in utterance (9) was incorrectly recognized as "*meeting*". As a result, the appointment's title was changed to "*meeting*", which was not intended by the user. Realizing the system's error, the user attempted

to correct the title in the last utterance. The Dialogue Manager recognized the user's intention correctly, and formulated a domain task description to send to the Calendar agent for resolving the appointment. However, a bug in the Dialogue Manager caused the appointment's time, 2 pm, to be written in an incorrect format, which caused the Calendar agent to crash.

- 1 User Find appointments for Friday.
Found appointments for Friday.
 - 2 SPA You have an appointment with Wayne about conference paper on Friday at 11 am.
 - 3 User Please move this appointment to Monday.
Please move this appointment to Monday.
 - 4 SPA Are you sure you want to move appointment about conference paper to Monday 11 am?
 - 5 User Change to Monday 2 pm.
Change to Monday two p.m.
 - 6 SPA Please specify new title or attendees.
 - 7 User Conference paper.
Conference paper.
 - 8 SPA Please specify new title or attendees.
 - 9 User Meeting's title is conference paper.
Meeting title is conference paper.
 - 10 SPA Appointment has been modified.
 - 11 User Show me the schedule for Monday at 2 pm.
Show me the scheduled for Monday.
 - 12 SPA You have an appointment with Wayne about conference paper on Monday at 2 pm.
 - 13 User Change title for 2 pm meeting.
Change title for two p.m. meeting.
-

Figure 6.3: An Example Scenario for Task (6)

The seven failures of task (10) and (11) were all due to incorrect task results, i.e. all messages (in the Inbox) were deleted/moved. Task (10) requires the deletion of messages about “*the war with Israel*”, while in task (11), messages about “*seminars*” must be moved to the Seminars folder. In one case, the problem was because subject (6) himself/herself requested: “*Please delete all e-mails*” when viewing the messages about the war with Israel, and still did not realize his/her mistake when confirming the deletion, presumably thinking the domain of quantification to be the set of messages on the screen. The problem could have been avoided with a clearer request for confirmation. In other cases, similar misinterpretations occurred due to speech recognition errors or secondary errors made by the Partial Parser or the Dialogue Manager’s semantic analysis process. Several example dialogue excerpts between users and the SPA in performing task (10) and (11) are presented in Figure 6.4, which illustrate some of the cases. In this figure, speech recognition outputs are shown in italics. In the example in Figure 6.4(a), the speech recognition software misrecognized “*with subject Israel*”, consequently, the Partial Parser ignored “*be subject Israel*”. As a result, the Dialogue Manager misinterpreted subject (3)’s request as to *delete all e-mails* in the user’s Inbox. The user did not realize the problem, hence confirmed the deletion. Again, a clearer request for confirmation would have avoided the problem.

User Delete all e-mails with subject Israel.
 Delete all e-mails be subject Israel.

SPA Are you sure you want to delete those messages?

User Yes.
 Yes.

SPA Messages have been deleted.

(a) Speech Recognition and Parsing Errors

Figure 6.4: Example Failures of Task (10) and (11)

Considering another example, Figure 6.4(b), subject (9) used the reference “*all three messages*” to refer to the messages in the current context but the Dialogue Manager received only “*messages*” from the Partial Parser, thus the task objects were resolved to all e-mail messages in the Inbox (i.e. the default folder).

User Show all messages about seminars.
Show all messages about the seminar.

SPA You have 3 messages about seminar, 1 from Bruno Gaeta and 2 from Bernhard Hengst.

User Move all three messages to seminars folder.
Moved Wamsley messages to seminars folder.

SPA The messages have been moved to seminars folder.

(b) Speech Recognition Error

User Find e-mails about Israel.
Fined e-mails about Israel.

SPA You have 4 messages about Israel .

User Delete all these e-mails.
Delete all these e-mails.

SPA Are you sure you want to delete those messages?

User Yes.
Yes.

SPA Messages have been deleted.

(c) Speech Recognition, Parsing and User Errors

Figure 6.4: Example Failures of Task (10) and (11)

In the last example, Figure 6.4(c), an error in the semantic analysis stage of dialogue processing caused the Dialogue Manager’s misinterpretation of “*all these e-mails*” as an ALL-quantifier phrase. When the user uses an ALL-phrase and does not refer to objects in the current context (e.g. “*these e-mails*”), the system considers the user’s request as opening a new context. Thus user (7)’s request was incorrectly recognized as a new request to delete all e-mail messages.

The failures of task (10) and (11) suggest that due to speech recognition performance, users may not fully understand the system’s current context. Thus better confirmation requests by the SPA are needed. For example, including the number of e-mail messages in the requests for confirmation of deletion, e.g. “*Are you sure you want to delete those 15 messages*”, would help by providing grounding information about the system’s current context.

Considering the case of task (12), users are required to find and read an e-mail message from Kate sent today, in which the users are asked to reschedule an appointment with Kate to another time. However, there is another appointment with Mark at the same time and with the same title, making the task more difficult. Because the speech recognizer repeatedly failed to recognize “*Kate*” for user (5), the user had to give up the task, since he/she could not specify that the appointment to be rescheduled is the one with Kate, not with Mark. Subject (1) also gave up task (12) after a few utterances. The subject kept referring to the appointment using only the attendee’s name, “*Kate*”, but this kind of reference is not handled by the Dialogue Manager.

Overall, the evaluation has shown a high completion rate (88.33%) at the task-level. Although speech recognition performance was not perfect (86.94% concept-word recognition), the users managed to complete most of the tasks. Task failures were mainly due to speech recognition errors, in particular, incorrect recognition of proper names. In addition, some failures were because the users did not fully understand what features are supported by the system. If users were given more time than just the training sessions to explore the system, the task completion rate may have been higher.

Utterance-Level Performance

At the utterance-level, we evaluate the system performance by considering the appropriateness of the system’s responses. We distinguish two types of problem, which are not mutually exclusive: *unexpected* and *inappropriate* responses. Our definitions and explanations of these objective measures are presented in

Measure	Definition/Explanation
<i>Unexpected Responses</i>	System responses that are not expected by the user given the user's actual speech input and his/her interpretation of the current conversational context. For example, if the user requests " <i>Show my Inbox folder</i> ", from the user's point of view, any responses other than the system displaying the user's Inbox folder are unexpected. Thus if the speech recognizer misrecognizes "Inbox" as "Friends" and the system responds " <i>You don't have a folder named Friends</i> ", then the response is considered unexpected.
<i>Inappropriate Responses</i>	Considering the speech recognizer as an external component, given the speech recognition output as input to the course of dialogue processing, a system response is considered inappropriate if: <ul style="list-style-type: none"> - the user's intention could not be recognized. - the user's intention was incorrectly recognized. - the back-end application failed to perform the task. - the task was performed incorrectly.
<i>Recovery</i>	The Dialogue Manager recovers from problems or errors by appropriately requesting the user for clarification.

Table 6.3: Utterance-Level Evaluation Measures

Table 6.3. Note that the system is required to respond to every user utterance. In addition, we assume that the user's expected system response can be predicted by considering the actual speech input and is as determined by the experimenters. The sources of unexpected and inappropriate responses can be any system components and can also be because of the subject himself/herself. A course of dialogue processing is done in several stages, starting from the user, then the speech recognizer, the Partial Parser, the Dialogue Manager, and finally the back-end applications. We consider the first component (including the user) that causes the problem to be the source of the problem. One measure of system

performance at this level is based on the statistical distribution of the problem sources. Moreover, we are interested in how error recovery has been done by the Dialogue Manager. The objective evaluation at the utterance-level helps us understand how individual component performance contributes to the overall system performance and to identify additional issues with the SPA's dialogue model.

Tables C.2 in Section 3, Appendix C reports the evaluation results for each subject, including: the total number of spoken utterances, the number of utterances whose concepts are fully and correctly recognized by the speech recognizer, the number of utterances with some concepts not recognized, and the number of the system's unexpected responses, inappropriate responses and corresponding sources of problems. The distribution of problem sources causing unexpected and inappropriate responses is shown in Figure 6.5. Unexpected responses were caused by the user, the speech recognizer, the Partial Parser or the Dialogue Manager. Note that unexpected responses cannot be due to failures at the back-end domain applications because users do not know what e-mail and calendar data is used at the back-end, thus cannot predict if the task results are correct or not. The evaluation results presented in Figure 6.5(a) shows that most unexpected responses were due to speech recognition errors. Ignoring these speech recognition errors, the Dialogue Manager is the main source of inappropriate responses. This is expected because it is very unlikely that the back-end applications perform tasks incorrectly. In Tables C.2 in Appendix C, we also classify unexpected and inappropriate responses caused by the Dialogue Manager according to different dialogue management aspects. For a more detailed explanation, we now describe some specific examples obtained from the evaluation.

An example of an unexpected response is utterance (6) in the dialogue excerpt in Figure 6.6. This is unusual in that it is caused by the user. In utterance (1), user (9) requested to schedule a new appointment with Kathy. Because of an error made by the Partial Parser, the Dialogue Manager did not receive the

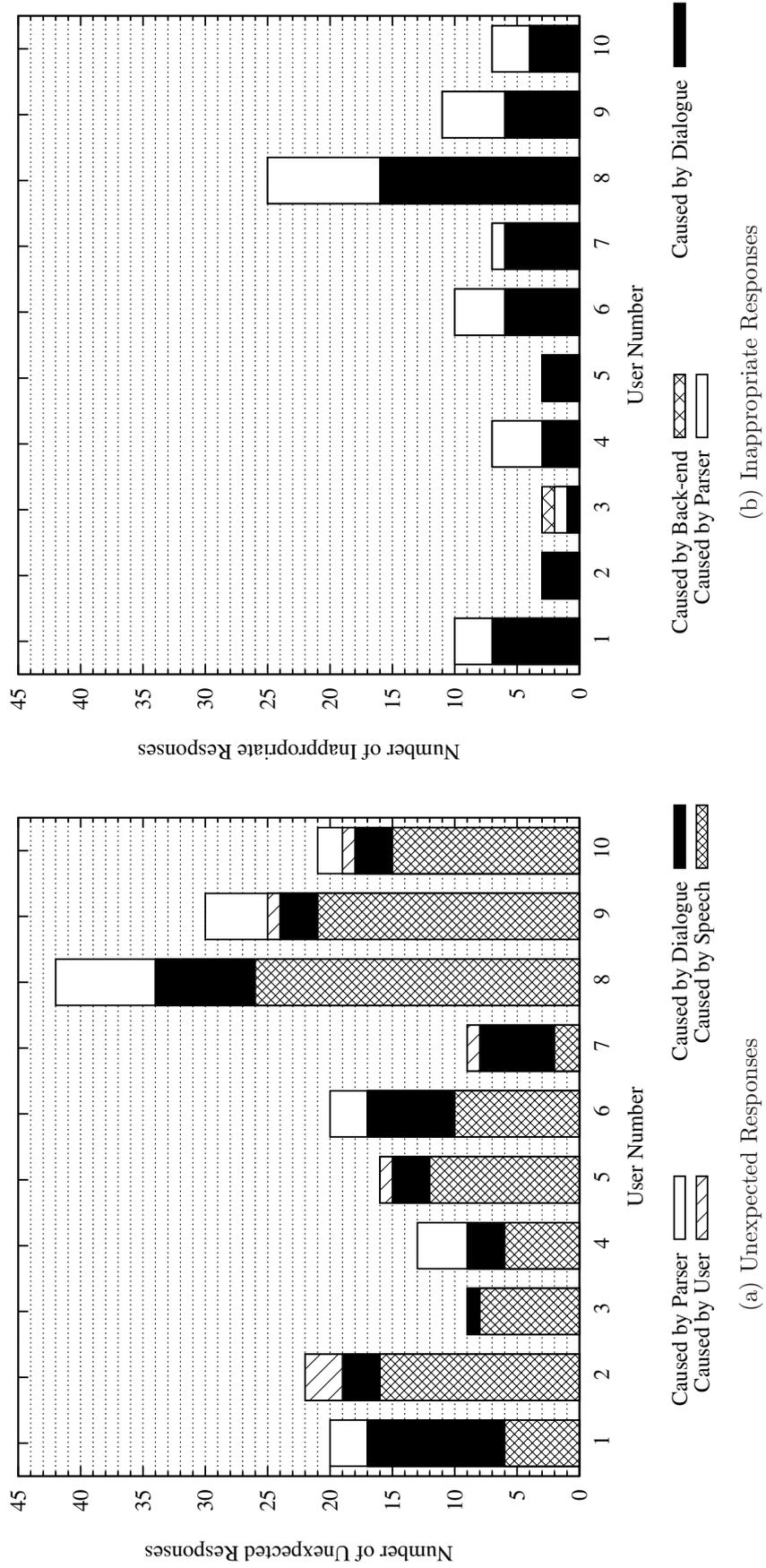


Figure 6.5: Source of Unexpected and Inappropriate Responses

information about the attendee, Kathy, which the system then asked the user to specify in utterance (2). However, the speech recognizer misrecognized the user's answer. Thus in utterance (5), it seemed that the user wanted to correct the attendee's name. However, because the user's request was not clear, he/she received an unexpected response, utterance (6), from the system.

- | | | |
|---|------|---|
| 1 | User | Schedule meeting with Kathy tomorrow at 11 am.
<i>Scheduled meeting with Kathy tomorrow at eleven a.m.</i> |
| 2 | SPA | Who will you meet? |
| 3 | User | Kathy.
<i>The.</i> |
| 4 | SPA | Appointment has been created for tomorrow 11 am. |
| 5 | User | 11 am meeting with Kathy.
<i>eleven a.m. meeting with Kathy.</i> |
| 6 | SPA | Do you want to create that appointment? |
-

Figure 6.6: Unexpected Response Caused by the User

Unexpected responses are also considered to be caused by users when they did not pronounce folder names, e.g. “*Seminars*”, exactly as specified in the task description. For example, some users received the response: “*Folder name Seminar not found*”.

The user's utterance is first processed by the speech recognizer to produce a text string. The Partial Parser then uses pattern matching techniques to recognize domain-specific keywords and extract important concept phrases from this speech recognition output. Correct recognition of the user's intention relies heavily on these concepts being extracted correctly. Thus errors made by the speech recognizer and the Partial Parser are likely to cause further incorrect dialogue processing, resulting in unexpected and/or inappropriate system responses. The most common error caused by speech recognition is not recognizing domain-specific keywords. For instance, in one case, the user's actual speech input, “*Show me e-mail from John Lloyd*”, was recognized as “*Show me*

in malls from John Noye". The system's response, which is not expected by the user, was "*Which one should I show?*". Another speech recognition error in this case is that the proper name, "*John Lloyd*", was not correctly recognized. In fact, incorrect speech recognition of proper names did cause many unexpected responses. In Figure 6.7, we show an example dialogue containing an inappropriate response caused by the Partial Parser, which is also unexpected from the user's point of view. In this example, user (6) used the temporal phrase "*3 pm*" before the keyword "*appointment*" to specify the meeting to be cancelled. In fact, this way of using temporal expressions was unique amongst our subjects. The temporal information, 3 pm, was lost due to the Partial Parser. As a result, the Dialogue Manager could not determine which appointment the user wanted to delete. In this case, the system's response, utterance (4), is inappropriate and also unexpected.

1	User	Delete. <i>Delete.</i>
2	SPA	Which one should I cancel?
3	User	3 pm appointment. <i>three p.m. appointment.</i>
4	SPA	Which appointment?

Figure 6.7: Unexpected/Inappropriate Response Caused by the Partial Parser

There was only one instance of an inappropriate response caused by the back-end, which occurred with user (3). The subject requested e-mail messages about "*the project*". In processing this request, the stop-word "*the*" was first discarded, next the E-mail agent called the back-end e-mail management software to search for messages that contain the string "*_project*". A single space was prepended to the string to work around a problem that the e-mail application does not detect word boundaries. However, as a result, the message with subject "*Re:project proposal*" was not found.

To understand in detail how performance of the Dialogue Manager affects the overall system performance, we attribute the cause of unexpected and inappropriate responses to different dialogue management aspects, which include semantic analysis, task (type) recognition, object determination, attribute extraction, reference resolution, resolution of temporal expression, dialogue strategy, clarification request generation and context tracking. In Table 6.4, we show the overall result for all users (the detailed results for individual users are presented in Tables C.2 in Section 3, Appendix C).

Total Number of Responses	569			
Dialogue Management Aspect	Number of Unexpected Responses		Number of Inappropriate Responses	
		48	(8.40%)	55
<i>Semantic Analysis</i>	3	(6.25%)	4	(7.27%)
<i>Context Tracking / Act Type Determination</i>	2	(4.16%)	2	(3.63%)
<i>Task Type Recognition</i>	7	(14.58%)	7	(12.72%)
<i>Object Determination</i>	13	(27.08%)	15	(27.27%)
<i>Object Attribute Extraction</i>	8	(16.66%)	17	(30.9%)
<i>Reference Resolution</i>	5	(10.41%)	4	(7.27%)
<i>Temporal Expression Resolution</i>	5	(10.41%)	6	(10.90%)
<i>Dialogue Strategy</i>	4	(8.33%)	0	(0%)
<i>Clarification Request Generation</i>	1	(2.08%)	0	(0%)

Table 6.4: Performance Distribution for Different Dialogue Management Aspects

Most unexpected and inappropriate responses were because the Dialogue Manager failed to extract attribute phrases and determine the corresponding objects. The reason for this is that we expected the use of keywords such as “*about*”, “*with*”, “*from*” for specifying appointment titles, appointment attendees, e-mail message senders, etc., for example, “*The e-mail message from Paul*” or “*The appointment about conference paper*”. Because speech recognition cannot be 100% accurate, the presence of these keywords is used in extracting other

unrestricted phrases. However, during the training the users were given only brief suggestions about what keywords to use. Thus the test sessions showed a variety of constructions for referring to appointments and e-mail messages, the most common being to refer to appointments using only the title (e.g. “*conference paper*”), such as in the example dialogue of user (8) in Figure 6.8. In addition, this user also tended to use “*regarding*” (instead of “*about*”) to indicate the title. In these cases, the Dialogue Manager was not able to extract the object’s attributes correctly.

-
- | | | |
|---|------|---|
| 1 | User | Move my appointment regarding conference paper.
<i>Move my appointment regarding conference paper.</i> |
| 2 | SPA | Which appointment should be rescheduled? |
| 3 | User | Conference paper.
<i>Conference paper.</i> |
| 4 | SPA | Which appointment should be rescheduled? |
-

Figure 6.8: Response Problem Caused by the Dialogue Manager (a)

Another cause of unexpected responses was a dialogue strategy which requires the user’s specification of domain objects for every task request. This strategy was adopted to minimize failures caused by speech recognition errors. In the example in Figure 6.9, the user’s speech input “*Delete it*” was recognized as “*Deleted*”, which was a common problem. The Dialogue Manager could identify that the user requested a deletion task, however it is unclear whether it was a request to delete the appointment in the current context or a new request to delete some other appointments or e-mail messages. According to the chosen strategy, the Dialogue Manager does not assume that the requested deletion is of the current appointment, but instead asks the user for more information. Therefore we do not consider the system’s clarification request, utterance (4), an inappropriate response, but an unexpected response according to the user’s perspective.

-
- | | | |
|---|------|---|
| 1 | User | Check appointments for Tuesday next week at 3 pm.
<i>Check appointments were Tuesday next week at three p.m.</i> |
| 2 | SPA | You have an appointment with Doctor
about health check Tuesday next week at 3 pm. |
| 3 | User | Delete it.
<i>Deleted.</i> |
| 4 | SPA | Which one should I cancel? |
-

Figure 6.9: Response Problem Caused by the Dialogue Manager (b)

An example of an unexpected response caused by the Dialogue Manager's unclear clarification request is given in the dialogue excerpt in Figure 6.10. In utterance (3), user (1) requested to reschedule the appointment with Wayne to next Monday. However, due to speech recognition and parsing errors, concept-words "it", "to" and "Monday" were lost. As a result, the Dialogue Manager interpreted the user's utterance as a request to reschedule next week's appointments to another time. The system switched the context to a new *reschedule* task and suggested to the user to move the appointments separately. Note that this response, utterance (4), is also an unexpected response, but this is due to the speech recognizer and the Partial Parser, not the user or the Dialogue Manager. However, because of the system's unclear response, the user did not recognize that the context had changed. Thus he/she repeated the request, hence received another unexpected response, utterance (6). This unexpected response is therefore caused by the previously unclear clarification request, utterance (4), of the Dialogue Manager. The system should have phrased the response more appropriately to indicate a change in context, for example using the formulation "*You have 2 appointments next week, which one do you want to reschedule?*", making the response more clearly a clarification request.

In a number of cases, the recognized task-type of a user request was wrong because of the use of verbs not in the system's vocabulary and not defined in the domain task description, such as "*rename*", "*find*", "*check*", etc. User (8), an

-
- | | | |
|---|------|---|
| 1 | User | Check appointments for Friday at 11 am.
<i>Check appointments for Friday.</i> |
| 2 | SPA | You have an appointment with Wayne about conference paper on Friday at 11 am. |
| 3 | User | Move it to Monday next week.
<i>Move it to mundane next week.</i> |
| 4 | SPA | There are 2 appointments match your condition, please reschedule one appointment at a time. |
| 5 | User | Move it to Monday next week.
<i>Move it to Monday next week.</i> |
| 6 | SPA | Which appointment should be rescheduled? |
-

Figure 6.10: Response Problem Caused by the Dialogue Manager (c)

administrative staff member who has to deal with people in her job, tended to use long and polite sentences such as “*Could you please schedule it in for 1 pm tomorrow*”. Although the Partial Parser is designed to perform shallow parsing, aiming to discard redundant words, this kind of complex utterance is likely to cause task recognition errors.

An interesting case, in which the system responded unexpectedly and inappropriately, was because of problems in context tracking and switching, as in the dialogue excerpt from user (6) given in Figure 6.11. It seems that at some point in the dialogue, the system did not switch context correctly. Considering utterances (7) and (9), because the verb “*find*” is not in the Dialogue Manager’s vocabulary and task description, these utterances are recognized as responses to the system’s clarification requests, not as new search requests. Thus the Dialogue Manager considered these two utterances as contributing to the current context of the appointment deletion task. Perhaps the user did have the same intention in utterance (7), i.e. continuing the deletion. However, because the system could not understand the user³, it seemed that the user wanted to switch

³As discussed before, we expect the user to use important keywords such as “*about*”, “*from*”, etc. in referring to appointments and e-mail messages.

-
- | | | |
|----|------|---|
| 1 | User | Show me the schedule for Tuesday.
<i>Show me the schedule Tuesday.</i> |
| 2 | SPA | You have an appointment with Doctor
about health check on Tuesday at 3 pm. |
| 3 | User | Delete.
<i>Delete.</i> |
| 4 | SPA | Which one should I cancel? |
| 5 | User | Delete health check appointment.
<i>The leaked health check appointment.</i> |
| 6 | SPA | Which appointment? |
| 7 | User | Please find health check appointment.
<i>Please find health check appointment.</i> |
| 8 | SPA | Which appointment? |
| 9 | User | Please find appointment about health check.
<i>Please find appointment about health check.</i> |
| 10 | SPA | Are you sure you want to delete that appointment? |
| 11 | User | Yes.
<i>Yes.</i> |
| 12 | SPA | Appointment has been deleted. |
-

Figure 6.11: Response Problem Caused by the Dialogue Manager (d)

context in utterance (9), starting a new request to search for the appointment. Unexpectedly, the system identified the appointment and asked the user to confirm the deletion. So utterance (10) was both unexpected and inappropriate even though it led to the task being completed successfully. The main evidence for our interpretation is that the user seemed surprised when hearing utterance (10). However, it was not clear at the time of utterance (9), even to us, whether the user did indeed intend to switch contexts, and we can think of no method the SPA could use to determine this based only on the input available to the system.

The graphs in Figure 6.12 show the total number of clarification requests per user and per task. Table C.3 in Appendix C shows the utterance-level performance for each user, giving the total number of utterances taken by the user, the number of utterances that had all concepts recognized correctly by the speech recognizer, and the number of unexpected and inappropriate responses that the user received and their problem sources.

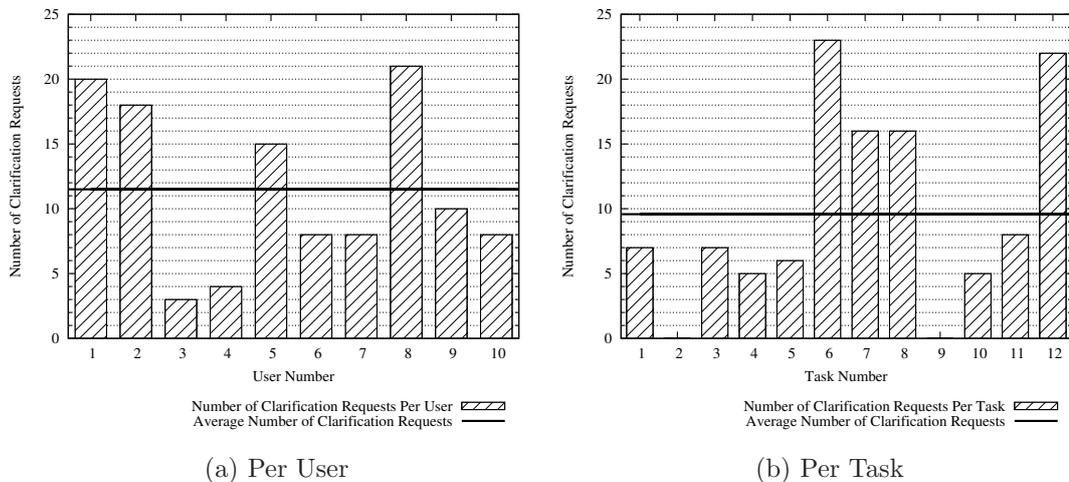


Figure 6.12: Number of Clarification Requests by the Dialogue Manager

The evaluation results show that the number of clarification requests required for the users is consistent with their concept-word speech recognition accuracy (Figure 6.1). Users (1), (2) and (8), whose speech recognition accuracy was below average, experienced more clarification requests from the system. An exceptional case is user (5), who had highly accurate speech recognition but still required an above average number of clarifications. The reason for this is that the speech recognizer could not recognize the proper name *Kate* for this user. However, the user must be able to use this proper name to complete task (12), to be able to distinguish an appointment with *Kate* from another appointment with *Mark* at the same time and with the same title. As the user could not correctly specify the appointment with *Kate*, the system had to request the user for clarification many times. The number of clarification requests generated per task also agrees with the complexity of the task. Task (6), (7) and (8), which are

more complex than the other tasks, required more clarifications. Similarly, the system generated more clarification requests in handling task (12), a complex combined e-mail and calendar task.

In addition, the results obtained from system logs show that the system's processing time (including dialogue and domain task processing) for a user input was less than 1 second, which is negligible in comparison with the time required for transferring data between the front-end device (the PDA) and the SPA server. The users experienced at some point a small delay due to ad-hoc wireless transmission. However, most of the time, the system's responses were delivered in a timely manner.

6.3.2 Subjective Results

The summary of the subjective evaluation results are reported in Table 6.5. Table C.1 in Appendix C shows the detailed ratings for each individual user. Overall, user comments and ratings are very positive. Most users were happy with the performance of the SPA and would like to use such a system in future. There are three users who thought that using the system was frustrating. This response is predictable as one of these users (subject (8)) suffered the worst speech recognition performance, and the other two had trouble from the very first task, which is the simplest task for all other users. For these two users, the speech recognizer failed many times to recognize the family name "*Lloyd*", thus it took them a large number of utterances to complete the task. The system certainly cannot recover from all kinds of speech recognition errors, thus it is understandable that the user's level of frustration increases as speech recognition performance gets worse. Moreover, as the user becomes more and more frustrated, the speech recognition performance deteriorates further, causing more frustration.

Our approach to dialogue management is aimed to allow users to interact effectively with an integrated collection of back-end domain assistants through

Question	Mean
<i>It was easy to learn how to use the SPA</i>	4.2
<i>It was easy to navigate through the SPA</i>	3.8
<i>Feedback from the SPA is clear and easy to understand</i>	4.3
<i>The SPA worked the way I expected it to</i>	3.7
<i>The SPA understood what I asked it to do a lot of the time</i>	4.0
<i>It took a lot of effort to remember how to perform tasks</i>	1.8
<i>It was easy to ask questions and make requests that the SPA could understand</i>	3.7
<i>The SPA gave reasonable responses to my questions and requests</i>	4.1
<i>The SPA gave appropriate responses when it didn't understand what I said</i>	4.0
<i>It was easy to recover from mistakes that I made</i>	3.5
<i>Using the SPA is frustrating</i>	2.9
<i>The SPA responded in a timely manner</i>	4.1
<i>The e-mail interface is clear and easy to understand</i>	4.3
<i>It was easy to find e-mail messages</i>	4.2
<i>It was easy to delete e-mail messages</i>	4.2
<i>It was easy to move e-mail messages</i>	4.1
<i>It was easy to move between e-mail and calendar in the SPA</i>	4.5
<i>The calendar interface is clear and easy to understand</i>	4.5
<i>It was easy to make appointments</i>	3.7
<i>It was easy to delete appointments</i>	4.3
<i>It was easy to change appointments</i>	3.5
<i>I was happy about the overall performance of the SPA</i>	4.1
<i>I would use a system like the SPA in future</i>	3.8

Table 6.5: Summary of the User Survey Ratings

a single point of contact. Using the SPA, the users were able to switch back and forth to perform tasks between the e-mail and calendar domains. We designed the evaluation so that users are required to perform a series of tasks in both domains. Moreover, task (12) is more complex, requiring users to combine information in these two domains. After the test sessions, all users either agreed or strongly agreed in the survey that it is easy to switch between e-mail and calendar using the SPA.

6.4 Comments

As we had expected, speech recognition could not be 100% accurate although the performance of the Dragon NaturallySpeaking software was extremely good, considering the experimental setup. The main problem was that of the speech recognizer not recognizing proper names, while in the e-mail and calendar management domains, this is essential to allow users to refer to their appointments and e-mail messages. A possible improvement could be to use a phonetic dictionary to match names with entries in the user's address book. Another way to overcome this problem could be to collect proper names in the user's e-mail and calendar data and add them to the user's speech profile to improve recognition accuracy, if this feature is supported by the speech recognition software. However, a drawback of these methods is that they cannot handle new names. A complementary solution could be supporting multimodal input so that the user can either highlight or type in proper names using the GUI. Note that multimodal support is available in the SPA. Nevertheless, for our user study to evaluate the SPA's dialogue model, the users were restricted to using only the speech modality in their requests. Apart from problems due to incorrectly recognized proper names, the SPA was able to overcome speech recognition errors in most cases, allowing the users to successfully complete most of the tasks.

Input	Output
Yes	<i>“yet this”, “year ’s”, “guess”, “gets”, “yass”, “gas”, “US”, “next”</i>
Yes please	<i>“yesterday ’s”, “yesterdays”, “this please”</i>
No	<i>“none”, “Narang”, “narrow”, “know”, “naga”</i>

Table 6.6: Example Speech Recognition Output of *Yes/No*

Another common speech recognition problem was the difficulty in recognizing simple words such as *“yes”* and *“no”* (Table 6.6), which is required in confirming the system's requests. It might be possible to bias the system towards yes/no in response to clarifications but this is not supported with Dragon NaturallySpeak-

ing. With multimodal support, users will be able to interact more effectively with the system, having the option of using the GUI to enter phrases that cannot be recognized by the speech software. In the user study, we noticed that some users occasionally used noun-noun compounds to refer to domain objects such as *“health check appointment”*. Allowing the use of such patterns can introduce more errors as it is more likely that the noun phrase title of the appointment is misrecognized by the speech recognizer. In addition, most users rarely used pronominal references to refer to objects and people. Pronouns such as *“him”*, *“them”*, *“it”*, etc., and ordinal numbers such as *“first”*, *“second”*, etc., were recognized correctly for all users almost all of the time. Thus they could make use of the conversational context to avoid using/repeating proper names and difficult phrases. We also observed that as the users became more familiar and confident with the system, they interacted more naturally, and started using more references.

Another issue with the dialogue model is tracking the conversational context. If users do not pay enough attention to the system’s responses, they may not notice that the context has been changed, moving forward according to the system’s interpretation of the user’s recent utterance. It might be possible that these changes are unexpected to the user due to significant speech recognition errors. As a result, the user and the system have different interpretations of the current context, which is likely to cause further misrecognition of the user’s intentions and more system misunderstanding of the user’s current context. This phenomenon is also common in human-human communication. In the case of the SPA, a solution could be to provide more information in the system’s confirmation and clarification requests so that users can recognize that the context has been changed. Our observation from the evaluation was that if users detect differences between their and the system’s context, most of them resolve the situation by introducing a new request (possibly restarting the task) to switch to the correct context.

Our experience was that some users were more flexible in interacting with the system, in being able to think of different ways to recover from errors. We allowed users to see the speech recognition output through a text area on the PDA screen. Some users deliberately altered their language patterns to overcome speech recognition errors. For example, a user with a speech impediment changed from using the verb “reschedule”⁴ to “move”. Another user, who had a problem with “e-mail”, after a while changed to use “message”. In addition, although the users were given roughly same amount of training, some of them gained better knowledge of what the system can and cannot do than others, regardless of whether or not they had an IT background. Apparently, users with an IT background performed the tasks slightly better but not significantly better. However, our observations suggested that the user’s everyday activity strongly affected their preferred way of talking to and interacting with the system. For instance, a subject who has to deal with a lot of people every day talked to the SPA in a polite manner, using complex utterances. Another female subject, a secretarial assistant whose job requires careful management of appointments, tended to use temporal expressions in referring to appointments. In contrast, an IT male student gave the system short requests, and moreover, used pronouns quite often.

6.5 Overall Remarks

In this user study, we were interested in evaluating the SPA’s dialogue model. Although speech recognition using Dragon NaturallySpeaking was not perfect (average concept-word recognition accuracy of 86.94%), the evaluation results gave a high task completion rate (88.3%) and user scores (10.1/12). The subjective evaluation results also showed that the users are satisfied with the overall performance of the system. These results demonstrated the effectiveness and robustness of dialogue management in the SPA, providing evidence that our

⁴For this user, “reschedule” was always recognized as “rees scheduler”.

agent-based approach allows the development of robust and sophisticated spoken dialogue systems. However, note that this user study was carried out in a quiet environment. Thus future work could be to evaluate the system in different settings. In addition, evaluation of the system with multimodal support could be another area for future work.

CONCLUSION

The research in this thesis concerns dialogue management in personal assistant applications that allow user-system interaction through a spoken natural language dialogue. Our application, the *Smart Personal Assistant* (SPA), is an integrated collection of personal assistants, each specializing in a particular task domain. Users can interact with the SPA through a variety of devices such as PDAs, desktops, etc. Spoken dialogue systems are particularly useful for mobile devices because effective interaction using only the traditional GUI is not possible due to the device's limited I/O and processing capabilities. Our current implementation of the SPA provides users with remote access to their e-mail and calendar, allowing them to perform common e-mail and calendar tasks using a speech interface and a graphical user interface on PDAs. The SPA was implemented as a multi-agent system, in which dialogue management is performed by the Dialogue Manager agent, acting as the central component for coordinating the task assistants as well as maintaining a coherent spoken dialogue with the user.

We proposed an agent-based approach to dialogue management which meets the requirements for developing multi-domain spoken dialogue systems such as the SPA. Most existing methods for dialogue management are suitable for simple and highly constrained tasks. Other work on modelling dialogue for complex task domains, however, requires a very complex domain model for problem solving, which is unnecessarily complicated for the SPA. Spoken dialogue in complex domains such as in the case of the SPA requires a high level of sophistication, which is mainly user-driven, but system initiative is also essential for clarifying

user requests or notifying the user of important events. Dialogue management in the SPA exhibits agent-like behaviour, requiring some degree of reactivity in order to fulfil the user's requests, and at the same time, pro-activeness such as for error recovery.

Our approach makes use of a BDI agent architecture for dialogue modelling and control. In addition, existing work in speech act theory and discourse analysis has been applied in modelling dialogue interaction, using the concepts of conversational acts and discourse segment intentions. We consider dialogue interaction as rational action, the production of goal-directed behaviour. Thus in our approach, the dialogue model is explicitly encoded in the Dialogue Manager's agent plans, each specifying the agent's actions for handling a particular discourse-level or domain-level dialogue goal. The dialogue control flow is derived automatically as the result of the BDI interpretation cycle. The conversational context, the domain-specific knowledge and the user model are maintained in the agent's internal beliefs. We have implemented the Dialogue Manager agent of the SPA using the JACK Intelligent AgentsTM a Java-based agent development framework derived from a BDI agent architecture.

The proposed BDI approach allows a high level of abstraction in designing and developing the dialogue model. The dialogue plans are complex but modular units, each associated with a communicative goal or dialogue aspect. Although individual plans might be complex, they are highly abstracted and the control flow is derived simply and automatically by defining the appropriate triggers for the plans. Our approach preserves the system modularity at the level of plans, allowing domain-independent discourse-level plans to be reused in other domains. In addition, plan-level modularity also facilitates the addition of new plans. This enables the reuse and extension of the dialogue model in other applications and the integration of new back-end assistants into the existing system. By reusing the initial implementation of the SPA, which had only the e-mail task assistant, we successfully extended the dialogue model to support an additional calendar management domain.

Using our agent-based approach, the problem of achieving dialogue adaptation can be considered the problem of learning which plan to select and execute according to the current context. We proposed in this research a general method for incorporating learning into the BDI agent architecture. The learning process is done as the agent's meta-reasoning for learning plan selection strategies. We extended the SPA's Dialogue Manager to produce a plan-based dialogue agent capable of generate adaptive dialogue responses according to the conversational context and the user's device and preferences.

The strong potential of our agent-based approach for developing multi-domain personal assistant applications has been clearly demonstrated through the evaluation results, which shows a high task completion rate. We have carried out user studies of the SPA with the intention of evaluating the robustness and effectiveness of the SPA's dialogue model. Although speech recognition accuracy was not perfect, the system was able to recover from speech recognition errors, allowing the users to complete most of the evaluation tasks successfully. In addition, most users were satisfied with the overall performance of the system and indicated their willingness to use such a system in future.

7.1 Future Research

There are a number of areas of improvement for the current SPA system that might be undertaken as future work. In Chapter 5, we described our work in integrating learning into the agent-based dialogue model for achieving dialogue adaptation. However, we have only considered one learning problem, learning to generate adaptive system responses. There are other kinds of adaptivity that can improve the system usability, such as learning adaptive dialogue strategies according to the conversational context and the user's preferences. In addition, learning the user's language usage patterns, such as the use of references (referring to people using nicknames, etc.), can help in recognizing the user's requests. Our current method to obtain the user's preferences is to explicitly ask him/her

for the most preferred option. However, the user's preferences might be learned by considering the user's course of interaction.

Our evaluation results of the SPA showed that speech recognition errors can cause misinterpretation of the user's requests, and as a result, the user's misunderstanding about the current context. This problem can be avoided by better dialogue strategies, such as providing clearer clarification and confirmation requests. In addition, the dialogue model can be improved to be able to handle more commonly used language patterns. This can be done by improving the domain-specific vocabulary and domain-level dialogue plans. In Chapter 6, we discussed several solutions to overcoming common speech recognition problems, particularly for the recognition of proper names, including the use of a phonetic dictionary, biasing the recognition towards names in the user's address book, allowing the user to add names to the vocabulary of the speech recognition system dynamically, and allowing multimodal input for the user to select proper names from the graphical interface.

Finally, our work also suggests possibilities for future research in dialogue management. Current trends show an increased use of mobile devices for different types of services such as mobile banking, information seeking, etc. Spoken dialogue interaction would be useful for these applications in order to improve usability. The dialogue model of the SPA could be reused and extended for use in these other domains. In addition, there are other potential dialogue applications which require multimodal support, such as vehicle control, direction and map services, etc. Thus multimodal dialogue management using our agent-based approach could be another future research problem. Moreover, supporting collaboration through a "team" of Smart Personal Assistants could also be an interesting research area that has not been given much consideration. Lastly, our method for incorporating learning into the BDI agent architecture can be applied to the general problem of learning in BDI agents.

BIBLIOGRAPHY

- Allen, J. F. (1979) *A Plan-Based Approach to Speech Act Recognition*, Ph.D. thesis, Department of Computer Science, University of Toronto.
- Allen, J. F., Blaylock, N. and Ferguson, G. (2002) A Problem Solving Model for Collaborative Agents, in *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 774–781.
- Allen, J. F., Byron, D., Dzikovska, M., Ferguson, G., Galescu, L. and Stent, A. (2000) An Architecture for a Generic Dialogue Shell, *Natural Language Engineering*, 6(3):1–16.
- Allen, J. F., Byron, D. and Dzilovska, M. (2001) Towards Conversational Human-Computer Interaction, *AI Magazine*, 22(4):27–37.
- Allen, J. F., Miller, B. W., Ringger, E. K. and Sikorski, T. (1996) A Robust System for Natural Spoken Dialogue, in *Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics (ACL'96)*, pages 62–70.
- Ardissono, L., Boella, G. and Lesmo, L. (2000) A Plan-Based Agent Architecture for Interpreting Natural Language Dialogue, *International Journal of Human-Computer Studies*, 52(4):583–635.
- Aust, H., Oerder, M., Seide, F. and Steinbiss, V. (1995) The Philips Automatic Train Timetable Information System, *Speech Communication*, 17(3–4):249–262.
- Austin, J. L. (1962) *How to Do Things with Words*, Clarendon Press, Oxford.

- Azvine, B., Djian, D., Tsui, K. C. and Wobcke, W. (2000) The Intelligent Assistant: An Overview, in B. Azvine, N. Azarmi and D. Nauck (eds.), *Intelligent Systems and Soft Computing*, pages 215–238, Springer-Verlag, Berlin.
- Bernsen, N. O., Dybkjær, H. and Dybkjær, L. (1998) *Designing Interactive Speech Systems: From First Ideas to User Testing*, Springer-Verlag, Berlin.
- Bilange, E. (1991) A Task Independent Oral Dialogue Model, in *Proceedings of the Fifth European Chapter Meeting of the Association for the Computational Linguistics*, pages 83–88.
- Billsus, D., Brunk, C. A., Evans, C., Gladish, B. and Pazzani, M. (2002) Adaptive Interfaces for Ubiquitous Web Access, *Communications of the ACM*, 45(5):34–38.
- Bohlin, P., Cooper, R., Engdahl, E. and Larsson, S. (1999) Information States and Dialogue Move Engines, in *Proceedings of the IJCAI'99 Workshop on Knowledge and Reasoning in Practical Dialogue Systems*, pages 25–31.
- Bos, J. and Gabsdil, M. (2000) First-Order Inference and the Interpretation of Questions and Answers, in *Proceedings of the Fourth Workshop on the Semantics and Pragmatics of Dialogue*, pages 43–50.
- Bos, J., Klein, E., Lemon, O. and Oka, T. (2003) DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture, in *Proceedings of the Fourth SIGdial Workshop on Discourse and Dialogue*, pages 115–124.
- Bratman, M. E. (1987) *Intentions, Plans, and Practical Reason*, Harvard University Press, Cambridge, MA.
- Bruce, B. C. and Newman, D. (1978) Interacting Plans, *Cognitive Science*, 2:195–233.
- Carlson, R. (1996) The Dialog Component in the Waxholm System, in *Proceedings of the Eleventh Twente Workshop on Language Technology*, pages 209–218.

Chin, D. N. (1991) Intelligent Interfaces as Agents, in J. W. Sullivan and S. W. Tyler (eds.), *Intelligent User Interfaces*, ACM Press, New York, NY.

Chu-Carroll, J. (2000) MIMIC: An Adaptive Mixed Initiative Spoken Dialogue System for Information Queries, in *Proceedings of the Sixth ACL Conference on Applied Natural Language Processing*, pages 97–104.

Churcher, G. E., Atwell, E. S. and Souter, C. (1997) *Dialogue Management Systems: A Survey and Overview*, Technical Report 97.06, School of Computing, University of Leeds.

Clark, H. H. (1996) *Using Language*, Cambridge University Press, Cambridge.

Cohen, P. (1981) The Need for Referent Identification as a Planned Action, in *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, pages 31–36.

Cohen, P. and Perrault, C. (1979) Elements of a Plan-Based Theory of Speech Acts, *Cognitive Science*, 3:177–212.

Corchado, J. M., Pavón, J., Corchado, E. and Castillo, L. F. (2004) Development of CBR-BDI Agents: A Tourist Guide Application, in P. Funk and P. A. G. Calero (eds.), *Advances in Case-Based Reasoning: 7th European Conference (ECCBR 2004)*, pages 547–559, Springer-Verlag, Berlin.

Dybkjær, L. and Bernsen, N. O. (2001) Usability Evaluation in Spoken Language Dialogue Systems, in *Proceedings of the ACL 2001 Workshop on Evaluation Methodologies for Language and Dialogue Systems*, pages 1–10.

Engel, R. and Pflieger, N. (2006) Modality Fusion, in W. Wahlster (ed.), *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 223–236, Springer-Verlag, Berlin.

Estival, D. (2002) The Syrinx Spoken Language System, *International Journal of Speech Technology*, 5(1):85–96.

Firby, R. J. (1989) *Adaptive Execution in Complex Dynamic Worlds*, Ph.D. thesis, Department of Computer Science, Yale University.

Fitzgerald, W. and Firby, R. J. (1998) The Dynamic Predictive Memory Architecture: Integrating Language with Task Execution, in *Proceedings of IEEE International Joint Symposia on Intelligence and Systems*, pages 202–206.

Fitzgerald, W. and Firby, R. J. (2000) Dialog is Task Execution; Task Execution is Best Done Reactively; Therefore, Dialog Systems Call for a Reactive Task Execution Architecture, in *Proceedings of the 2000 AAAI Spring Symposium Workshop on Natural Dialogues with Practical Robotic Devices*.

Frampton, M. and Lemon, O. (2006) Learning More Effective Dialogue Strategies Using Limited Dialogue Move Features, in *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics (COLING-ACL'06)*, pages 185–192.

Frøkjær, E., Hertzum, M. and Hornbæk, K. (2000) Measuring Usability: Are Effectiveness, Efficiency, and Satisfaction Really Correlated?, in *Proceedings of the SIGCHI'00 Conference on Human Factors in Computing Systems*, pages 345–352.

Georgeff, M. P. and Ingrand, F. F. (1989) Decision-Making in an Embedded Reasoning System, in *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 972–978.

Georgeff, M. P. and Lansky, A. L. (1987) Reactive Reasoning and Planning, in *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI'87)*, pages 677–682.

Grice, H. P. (1957) Meaning, *The Philosophical Review*, 66:377–388.

Grice, H. P. (1975) Logic and Conversation, in P. Cole and J. Morgan (eds.), *Syntax and Semantics*, Vol. 3, pages 43–58, Academic Press, New York, NY.

Grosz, B. J. (1977) The Representation and Use of Focus in Understanding Dialogs, in B. J. Grosz, K. Sparck Jones and B. L. Webber (eds.), *Readings in Natural Language Processing*, pages 56–67, Morgan Kaufmann, Los Altos, CA.

Grosz, B. J. and Sidner, C. (1986) Attention, Intentions, and the Structure of Discourse, *Computational Linguistics*, 12:175–204.

Grosz, B. J., Weinstein, S. and Joshi, A. K. (1995) Centering: A Framework for Modeling the Local Coherence of Discourse, *Computational Linguistics*, 21:203–225.

Guerra-Hernández, A., El Fallah-Seghrouchni, A. and Soldano, H. (2004) Learning in BDI Multi-agent Systems, in J. Dix and J. A. Leite (eds.), *Computational Logic in Multi-Agent Systems*, pages 218–233, Springer-Verlag, Berlin.

Herzog, G. and Reithinger, N. (2006) The SmartKom Architecture: A Framework for Multimodal Dialogue Systems, in W. Wahlster (ed.), *SmartKom: Foundations of Multimodal Dialogue Systems*, pages 55–70, Springer-Verlag, Berlin.

Ho, V., Wobcke, W. and Compton, P. (2003) EMMA: An E-Mail Management Assistant, in *Proceedings of the 2003 IEEE/WIC International Conference on Intelligent Agent Technology*, pages 67–74.

Horvitz, E. (1999) Principles of Mixed-Initiative User Interfaces, in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems*, pages 159–166.

Howden, N., Rönquist, R., Hodgson, A. and Lucas, A. (2001) JACK Intelligent AgentsTM– Summary of an Agent Infrastructure, *Paper presented at the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS*, Montreal.

Jokinen, K., Kerminen, A., Kaipainen, M., Jauhiainen, T., Wilcock, G., Turunen, M., Hakulinen, J., Kuusisto, J. and Lagus, K. (2002) Adaptive Dialogue

Systems - Interaction with Interact, in *Proceedings of the Third SIGdial Workshop on Discourse and Dialogue*, pages 64–73.

Jönsson, A. (1997) Model for Habitable and Efficient Dialogue Management for Natural Language Interaction, *Natural Language Engineering*, 3(2):103–122.

Kamp, H. (1981) A Theory of Truth and Semantic Representation, in J. Groenendijk, T. Janssen and M. Stokhof (eds.), *Formal Methods in the Study of Language: Part 1*, pages 277–322, Mathematisch Centrum, Amsterdam.

Larsen, L. B. and Baekgaard, A. (1994) Rapid Prototyping of a Dialogue System Using a Generic Dialogue Development Platform, in *Proceedings of the Third International Conference on Spoken Language Processing*, pages 919–922.

Larsson, S. and Traum, D. (2000) Information State and Dialogue Management in the TRINDI Dialogue Move Engine Toolkit, *Natural Language Engineering*, 6:323–340.

Lewin, I. (1998) *Autoroute Dialogue Demonstrator*, Technical Report CRC-073, SRI Cambridge.

Lewis, E. (1993) Interactive Speech in Computer-Aided Learning, in C. Baber and J. M. Noyes (eds.), *Interactive Speech Technology*, pages 37–44, Taylor & Francis, London.

Lloyd, J. W. (2003) *Logic for Learning: Learning Comprehensible Theories from Structured Data*, Springer-Verlag, Berlin.

LuperFoy, S., Loehr, D., Duff, D., Miller, K., Reeder, F. and Harper, L. (1998) An Architecture for Dialogue Management, Context Tracking, and Pragmatic Adaptation in Spoken Dialogue Systems, in *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL'98)*, pages 794–801.

- Maes, P. (1994) Agents that Reduce Work and Information Overload, *Communications of the ACM*, 37(7):31–40.
- Mann, W. and Thompson, S. (1988) Rhetorical Structure Theory: Toward a Functional Theory of Text Organization, *Text*, 8(3):243–281.
- Martin, D., Cheyer, A. and Moran, D. (1999) The Open Agent Architecture: A Framework for Building Distributed Software Systems, *Applied Artificial Intelligence*, 13(1–2):91–128.
- Matheson, C., Poesio, M. and Traum, D. (2000) Modelling Grounding and Discourse Obligations Using Update Rules, in *Proceedings of the First Conference on North American Chapter of the Association for Computational Linguistics*, pages 1–8.
- McGlashan, S. (1996) Towards Multimodal Dialogue Management, in *Proceedings of the Eleventh Twente Workshop on Language Technology*, pages 13–22.
- Mirkovic, D. and Cavedon, L. (2005) Practical Plug-and-Play Dialogue Management, in *Proceedings of the Sixth Meeting of the Pacific Association for Computational Linguistics*.
- Muggleton, S. (1995) Inverse Entailment and Progol, *New Generation Computing*, 13(3–4):245–286.
- Nguyen, A. and Wobcke, W. (2005) An Agent-Based Approach to Dialogue Management in Personal Assistants, in *Proceedings of the 2005 International Conference on Intelligent User Interfaces*, pages 137–144.
- Nguyen, A. and Wobcke, W. (2006a) An Adaptive Plan-Based Dialogue Agent: Integrating Learning into a BDI Architecture, in *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 786–788.

- Nguyen, A. and Wobcke, W. (2006b) Extensibility and Reuse in an Agent-Based Dialogue Model, in *Proceedings of the IAT'06 International Workshop on Communication Between Human and Artificial Agents*, pages 367–371.
- Novick, D. G. and Sutton, S. (1996) Building on Experience: Managing Spoken Interaction through Library Subdialogues, in *Proceedings of the Eleventh Twente Workshop on Language Technology*, pages 51–60.
- Olivia, C., Chang, C. F., Enguix, C. F. and Ghose, A. K. (1999) Case-Based BDI Agents: An Effective Approach for Intelligent Search on the World Wide Web, in *Intelligent Agents in Cyberspace: Papers from the 1999 AAAI Spring Symposium*, pages 20–27.
- Oviatt, S. (1999) Ten Myths of Multimodal Interaction, *Communications of the ACM*, 42(11):74–81.
- Paraiso, E. C. and Barthès, J.-P. A. (2005) SpeechPA: An Ontology-Based Speech Interface for Personal Assistants, in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 657–663.
- Peckham, J. (1991) Speech Understanding and Dialogue Over the Telephone: An Overview of the ESPRIT SUNDIAL Project, in *Proceedings of the Workshop on Speech and Natural Language*, pages 14–27.
- Pellom, B., Ward, W., Hansen, J., Hacioglu, K., Zhang, J., Yu, X. and Pradhan, S. (2001) University of Colorado Dialog Systems for Travel and Navigation, in *Proceedings of the 2001 Human Language Technology Conference*, pages 1–6.
- Perrault, C. R., Allen, J. F. and Cohen, P. R. (1978) Speech Acts as a Basis for Understanding Dialogue Coherence, in *Proceedings of the 2nd Conference on Theoretical Issues in Natural Language Processing*, pages 125–132.
- Phung, T., Winikoff, M. and Padgham, L. (2005) Learning within the BDI Framework: An Empirical Analysis, in R. Khosla, R. J. Howlett and L. C.

- Jain (eds.), *Knowledge-Based Intelligent Information and Engineering Systems, Part III*, pages 282–288.
- Potamianos, A., Ammicht, E. and Kuo, H.-K. J. (2000) Dialogue Management in the Bell Labs Communicator System, in *Proceedings of the Sixth International Conference on Spoken Language Processing*, pages 603–606.
- Power, R. (1979) The organisation of purposeful dialogues, *Linguistics*, 17:107–152.
- Prince, E. (1981) Toward a Taxonomy of Given-New Information, in P. Cole (ed.), *Radical Pragmatics*, pages 223–255, Academic Press, New York, NY.
- Quinlan, J. R. (1986) Induction of Decision Trees, *Machine Learning*, 1:81–106.
- Quinlan, J. R. (1990) Learning Logical Definitions from Relations, *Machine Learning*, 5:239–266.
- Rao, A. S. and Georgeff, M. P. (1995) BDI-Agents: From Theory to Practice, in *Proceedings of the First International Conference on Multiagent Systems*, pages 312–319.
- Rich, C. and Sidner, C. L. (1998) COLLAGEN: A Collaboration Manager for Software Interface Agents, *User Modeling and User-Adapted Interaction*, 8(3–4):315–350.
- Rich, C., Sidner, C. L. and Lesh, N. (2001) Collagen: Applying Collaborative Discourse Theory to Human-Computer Interaction, *AI Magazine*, 22(4):15–25.
- Rickel, J., Lesh, N., Rich, C., Sidner, C. L. and Gertner, A. (2000) Using a Model of Collaborative Dialogue to Teach Procedural Tasks, in *Proceedings of the Workshop on Tutorial Dialog Systems at the Tenth International Conference on Artificial Intelligence in Education*.
- Sadek, D. (1999) Design Considerations on Dialogue Systems: From Theory to Technology - The Case of Artemis, in *Proceedings of the ESCA Workshop on Interactive Dialogue in Multi-Modal Systems*, pages 173–187.

- Sammut, C. A. (2001) Managing Context in a Conversational Agent, *Electronic Transactions on Artificial Intelligence*, 5(B):189–202.
- Searle, J. R. (1969) *Speech Acts: An Essay in the Philosophy of Language*, Cambridge University Press, Cambridge.
- Seneff, S. and Polifroni, J. (2000) Dialogue Management in the Mercury Flight Reservation System, in *Proceedings of the ANLP/NAACL 2000 Workshop on Conversational Systems*, pages 11–16.
- Smith, R. W. and Hipp, D. R. (1994) *Spoken Natural Language Dialog Systems: A Practical Approach*, Oxford University Press, Oxford.
- Stibler, K. and Denny, J. (2001) A Three-Tiered Evaluation Approach for Interactive Spoken Dialogue Systems, in *Proceedings of the First International Conference on Human Language Technology*, pages 1–5.
- Strube, M. (1998) Never Look Back: An Alternative to Centering, in *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL '98)*, pages 1251–1257.
- Thompson, C. A., Göker, M. H. and Langley, P. (2004) A Personalized System for Conversational Recommendations, *Journal of Artificial Intelligence Research*, 21:393–428.
- Tidhar, G., Heinze, C. and Selvestrel, M. (1998) Flying Together: Modelling Air Mission Teams, *Applied Intelligence*, 8:195–218.
- Traum, D. and Larsson, S. (2003) The Information State Approach to Dialogue Management, in J. van Kuppevelt and R. W. Smith (eds.), *Current and New Directions in Discourse and Dialogue*, Chapter 15, Kluwer, Dordrecht.
- Turunen, M. and Hakulinen, J. (2000) Mailman - A Multilingual Speech-Only E-Mail Client Based on an Adaptive Speech Application Framework, in

Proceedings of the 2000 Workshop on Multi-Lingual Speech Communication, pages 7–12.

Turunen, M. and Hakulinen, J. (2001) Agent-Based Adaptive Interaction and Dialogue Management Architecture for Speech Applications, in *Proceedings of the Fourth International Conference on Text, Speech and Dialogue*, pages 357–364.

Wahlster, W. (ed.) (2006) *SmartKom: Foundations of Multimodal Dialogue Systems*, Springer-Verlag, Berlin.

Walker, M. A. (2000) An Application of Reinforcement Learning to Dialogue Strategy Selection in a Spoken Dialogue System for Email, *Journal of Artificial Intelligence Research*, 12:387–416.

Walker, M. A., Fromer, J. and Narayanan, S. (1998) Learning Optimal Dialogue Strategies: A Case Study of a Spoken Dialogue Agent for Email, in *Proceedings of the 17th International Conference on Computational Linguistics and the 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL'98)*, pages 1345–1351.

Walker, M. A., Litman, D. J., Kamm, C. A. and Abella, A. (1997) PARADISE: A Framework for Evaluating Spoken Dialogue Agents, in *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 271–280.

Wallis, P., Mitchard, H., Das, J. and O’Dea, D. (2001) Dialogue Modelling for a Conversational Agent, in M. Stumptner, D. Corbett and M. Brooks (eds.), *AI 2001: Advances in Artificial Intelligence*, pages 532–544, Springer-Verlag, Berlin.

Weizenbaum, J. (1966) ELIZA - A Computer Program For the Study of Natural Language Communication Between Man and Machine, *Communications of the ACM*, 9(1):36–45.

Williams, S. (1996) Dialogue Management in a Mixed-Initiative, Cooperative, Spoken Language System, in *Proceedings of the Eleventh Twente Workshop on Language Technology*, pages 199–208.

Wobcke, W., Ho, V., Nguyen, A. and Krzywicki, A. (2005) A BDI Agent Architecture for Dialogue Modelling and Coordination in a Smart Personal Assistant, in *Proceedings of the 2005 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 323–329.

Wooldridge, M. J. and Jennings, N. R. (1995) Intelligent Agents: Theory and Practice, *The Knowledge Engineering Review*, 10(2):115–152.

Xu, W. and Rudnicky, A. (2000) Task-Based Dialog Management Using an Agenda, in *Proceedings of the ANLP/NAACL 2000 Workshop on Conversational Systems*, pages 42–47.

DIALOGUE MANAGEMENT

A.1 Example ProBot Scripts

```
-- recognizing utterance type

<wh-type> ==> [
  :Wh Question
  #assert(type(':WH_QUESTION'))
  #goto(type_supplement, ^0)
]

<yesno-type> ==> [
  :YesNo Question
  #assert(type(':YESNO_QUESTION'))
  #goto(type_supplement, ^0)
]

<request-type> ==> [
  :Request
  #assert(type(':IMPERATIVE'))
  #append(subject('you'))
  #goto(predicate, ^0)
]

<noun-phrase> ==> [
  :Declaration 1
  #assert(type(':DECLARATION'))
  #append(subject(^1))
  #print_result([])
]

<aff> ==> [
  :Declaration 2
  #assert(type(':DECLARATION'))
  #assert(type_supplement('AFF'))
  #print_result([])
]
```

```

<neg> ==> [
  :Declaration 3
  #assert(type('DECLARATION'))
  #assert(type_supplement('NEG'))
  #print_result([])
]

-- temporal expressions
day-type-phrase :: {the next <day-type> | the following <day-type> |
  the last <day-type> | the previous <day-type> | previous <day-type>
  |this <day-type> | next <day-type> | last <day-type> | <day-type>};

hour-type :: { hour | minute | second | pm | am | o'clock };

hour-type-phrase :: { <cardinal-numeral> <cardinal-numeral>
  <hour-type> | at <cardinal-numeral> <hour-type> |
  <cardinal-numeral> <hour-type> | <cardinal-numeral>
  <cardinal-numeral> | at <cardinal-numeral>};

date-type-phrase :: { <ordinal-numeral> | the <ordinal-numeral> };

other-time-type :: { now | right now | later | soon };

month-type :: { Januaray | February | March | April | May | June |
  July | August | September | October | November | December | month |
  months };

month-type-phrase :: {<ordinal-numeral> <month-type> | <month-type>};

time-phrase1 :: { <day-type-phrase> | <date-type-phrase> |
  <hour-type-phrase> | <month-type-phrase> | <other-time-type> };

time-phrase2 :: { at <time-phrase1> | on <time-phrase1> };

time-phrase :: { <time-phrase2> <time-phrase> | <time-phrase1>
  <time-phrase> | <time-phrase1> | <time-phrase2> };

appt-time-phrase :: { to <time-phrase> | for <time-phrase> |
  before <time-phrase> | after <time-phrase> | <time-phrase>
  forward~ | forward <time-phrase> | <time-phrase> backward~ |
  back <time-phrase> | backward <time-phrase> | <time-phrase> };

-- recognizing temporal expression for calendar domain

* <appt-time-phrase> ==> [
  #append(complement(^2))
  #append(direct_object(^1))
  #print_result([])
]

```

```

<appt-time-phrase> <noun-phrase> * ==> [
  #append(complement(^1))
  #append(direct_object(^2))
  #goto(indirect_object,^3)
]

* <appt-time-phrase> <noun-phrase> * ==> [
  #append(complement(^2))
  #append(direct_object(^1))
  #append(indirect_object(^3))
  #goto(indirect_object,^4)
]

* <appt-time-phrase> * ==> [
  #append(complement(^2))
  #append(direct_object(^1))
  #goto(indirect_object,^3)
]

<appt-time-phrase> #have(direct_object) ==> [
  #append(complement(^1))
  #print_result([])
]

<appt-time-phrase> * #have(direct_object) ==> [
  #append(complement(^1))
  #goto(indirect_object, ^2)
]

<appt-time-phrase> ==> [
  #append(complement(^1))
  #print_result([])
]

<appt-time-phrase> * #have(direct_object) ==> [
  #append(indirect_object(^2))
  #append(complement(^1))
  #print_result([])
]

<appt-time-phrase> * about * ==> [
  #append(direct_object(^2))
  #append(complement(^1))
  #goto(indirect_object, ['about'| ^3])
]

```

DIALOGUE ADAPTATION

B.1 Alkemy Learning Problem Specification

```
%% -- Data constructors for Email domain
-- the user's current device
PDA, DESKTOP : Device ;

-- the user's current modality
SPEECH, TEXT, NONE : Mode ;

-- the possible e-mail tasks
SEARCH, FIND, READ, SHOW, SUMMARIZE, REPLY, COMPOSE, SEND, FORWARD,
ARCHIVE, DELETE, NOTIFY : Task ;

-- the priority of the e-mail message
LOW, HIGH, NORMAL : Priority ;

-- the full sender name
Sender = String ;

-- length of the email (number of lines)
Length = Int ;

-- the folder name (virtual folder)
Folder = String ;

-- each message represented by its sender, length, folder and priority
Email = Sender * Length * Folder * Priority ;

-- possible Response Generation plans
ReturnResponse, ReturnMessageContent, ReturnMessageSummary,
ReturnMessageList, ReturnSubList, ReturnMessagesSortedBySender,
ReturnMessagesSortedByFolder, ReturnMessagesSortedByPriority : Plan ;

-- each individual case contains information about the device,
-- task, modality, the result set of e-mail messages and
-- the plan name
Individual = Device * Task * Mode * (Set Email) * Plan ;
```

B.1 Alkemy Learning Problem Specification

```
-- classes to be learned
True, False : Class ;

-- learn which Response Generation plan to be used
LEARN ResponsePlan : Individual -> Class ;

%% -- Initial training examples

ResponsePlan(PDA, SEARCH, SPEECH, {}, ReturnResponse) = True ;
ResponsePlan(PDA, SEARCH, SPEECH, {}, ReturnMessageContent) = False ;
ResponsePlan(PDA, SEARCH, SPEECH, {}, ReturnMessageSummary) = False ;

%% -- Transformations

-- projections to get information about the device, task, modality,
-- e-mail set and the plan name
projDevice : Individual -> Device ;
projDevice = project(0) ;

projTask : Individual -> Task ;
projTask = project(1) ;

projMode : Individual -> Mode ;
projMode = project(2) ;

projEmails : Individual -> (Set Email) ;
projEmails = project(3) ;

projPlan : Individual -> Plan ;
projPlan = project(4) ;

-- projections for the e-mail set to get sender set, folder set,
-- priority set (distinct priority values), priority list
projSetSender : (Set Email) -> (Set String) ;
projSetSender = projectSet(0) ;

projSetFolder : (Set Email) -> (Set String) ;
projSetFolder = projectSet(2) ;

projSetPriority : (Set Email) -> (Set Priority) ;
projSetPriority = projectSet(3) ;

projListPriority : (Set Email) -> (List Priority) ;
projListPriority = projectList(3) ;

-- projections for each e-mail message
projSender : Email -> String ;
projSender = project(0) ;
```

B.1 Alkemy Learning Problem Specification

```
projLength : Email -> Int ;
projLength = project(1) ;

projFolder : Email -> String ;
projFolder = project(2) ;

projPriority : Email -> Priority ;
projPriority = project(3) ;

-- calculate size of set
emailTrue : Email -> Bool ;
emailTrue = top() ;

numOfEmails : (Email -> Bool) -> (Set Email) -> Int ;
numOfEmails = domcard() ;

folderTrue : Folder -> Bool ;
folderTrue = top() ;

-- number comparison transformations
eq0 : Int -> Bool ;
eq0 = equality(0) ;

eq1 : Int -> Bool ;
eq1 = equality(1) ;

gt1 : Int -> Bool ;
gt1 = gt(1) ;

lt5 : Int -> Bool ;
lt5 = lt(5) ;

lt15 : Int -> Bool ;
lt15 = lt(15) ;

lt30 : Int -> Bool ;
lt30 = lt(30) ;

lt50 : Int -> Bool ;
lt50 = lt(50) ;

gte50 : Int -> Bool ;
gte50 = gte(50) ;

-- if then else
ite : (alpha -> Bool) -> (alpha -> Bool) -> (alpha -> Bool) ->
      alpha -> Bool ;
ite = ite() :: symmetric;
```

B.1 Alkemy Learning Problem Specification

```
-- conjunctions
and2 : (Individual -> Bool) -> (Individual -> Bool) ->
      Individual -> Bool ;
and2 = conjunction(2) ;

and3 : (Individual -> Bool) -> (Individual -> Bool) ->
      (Individual -> Bool) -> Individual -> Bool ;
and3 = conjunction(3) ;

or2 : (Individual -> Bool) -> (Individual -> Bool) ->
      Individual -> Bool ;
or2 = disjunction(2) ;

not : Bool -> Bool ;
not = negation() ;

true : alpha -> Bool ;
true = top() ;

none : Individual -> Bool ;
none = top() ;

bottom : Individual -> Bool ;
bottom = bottom() ;

-- check if some e-mail message exists
setEmailExists1 : (Email -> Bool) -> (Set Email) -> Bool ;
setEmailExists1 = setexists(1) ;

setDomCard : (alpha -> Bool) -> (Set alpha) -> Int ;
setDomCard = domcard() ;

listDomCard : (alpha -> Bool) -> (List alpha) -> Int ;
listDomCard = domcard() ;

oneEmail : alpha -> Bool ;
oneEmail = top() ;

moreThanOneEmail : alpha -> Bool ;
moreThanOneEmail = top() ;

-- default top
top : alpha -> Bool ;
top = top() ;

%% -- Rewrites

top >-> projTask . top ;           -- the task
top >-> projDevice . top ;        -- the device
```

B.1 Alkemy Learning Problem Specification

```
top >-> projMode . top ;                -- the mode

top >-> and2 (projEmails . numOfEmails (emailTrue) . eq0) (top) ;

top >-> and2 (projEmails . numOfEmails (emailTrue) . eq1)
           (projEmails . setEmailExists1 (projLength . gte50)) ;

top >-> and2 (projEmails.numOfEmails (emailTrue).eq1)
           (projEmails.setEmailExists1(projPriority.eqPriorityHIGH)) ;

top >-> projSetPriority . setDomCard (true) . top ;
top >-> projSetFolder . setDomCard (true) . top ;
top >-> projSetSender . setDomCard (true) . top ;
top >-> projEmails . numOfEmails (emailTrue) . top ;

top >-> projPlan . top ;                 -- the plan used

top >-> eqDevicePDA ;                   -- the device
top >-> eqDeviceDESKTOP ;

top >-> eqTaskSEARCH ;                 -- the task
top >-> eqTaskFIND ;
top >-> eqTaskREAD ;
top >-> eqTaskSHOW ;
top >-> eqTaskSUMMARIZE ;
top >-> eqTaskREPLY ;
top >-> eqTaskCOMPOSE ;
top >-> eqTaskSEND ;
top >-> eqTaskFORWARD ;
top >-> eqTaskARCHIVE ;
top >-> eqTaskDELETE ;
top >-> eqTaskNOTIFY ;

top >-> eqModeSPEECH ;                 -- the modality
top >-> eqModeNONE ;
top >-> eqModeTEXT ;

top >-> eqPriorityHIGH ;                -- the message priority
top >-> eqPriorityLOW ;
top >-> eqPriorityNORMAL ;

top >-> eqPlanReturnMessageContent ;    -- the plan name
top >-> eqPlanReturnMessageSummary ;
top >-> eqPlanReturnMessageList ;
top >-> eqPlanReturnMessagesSortedBySender ;
top >-> eqPlanReturnMessagesSortedByFolder ;
top >-> eqPlanReturnMessagesSortedByPriority ;
top >-> eqPlanReturnSubList ;
top >-> eqPlanReturnResponse ;
```

```
top >-> gte50 ;           -- numeric comparison
top >-> lt50 ;
top >-> lt30 ;
top >-> lt15 ;
top >-> lt5 ;
top >-> gt1 ;
top >-> eq1 ;
top >-> eq0 ;
lt50 >-> lt30 ;
lt30 >-> lt15 ;
lt15 >-> lt5 ;
lt5 >-> eq1 ;
lt5 >-> eq0 ;
gt1 >-> gte50 ;
%%
```

B.2 Alkemy Training Examples

```
-- Initial examples
ResponsePlan (PDA, SEARCH, SPEECH, {}, ReturnResponse) = True ;
ResponsePlan (PDA, SEARCH, SPEECH, {}, ReturnMessageList) = True ;
ResponsePlan (PDA, SEARCH, SPEECH, {}, ReturnMessageContent) = True ;
ResponsePlan (PDA, SEARCH, SPEECH, {}, ReturnMessageSummary) = True ;
ResponsePlan (PDA, SEARCH, SPEECH, {}, ReturnSubList) = True ;
ResponsePlan (PDA, SEARCH, SPEECH, {},
              ReturnMessagesSortedByFolder) = True ;
ResponsePlan (PDA, SEARCH, SPEECH, {},
              ReturnMessagesSortedBySender) = True ;
ResponsePlan (PDA, SEARCH, SPEECH, {},
              ReturnMessagesSortedByPriority) = True ;

ResponsePlan (PDA, SEARCH, TEXT, {}, ReturnResponse) = True ;
ResponsePlan (PDA, SEARCH, TEXT, {}, ReturnMessageList) = True ;
ResponsePlan (PDA, SEARCH, TEXT, {}, ReturnMessageContent) = True ;
ResponsePlan (PDA, SEARCH, TEXT, {}, ReturnMessageSummary) = True ;
ResponsePlan (PDA, SEARCH, TEXT, {}, ReturnSubList) = True ;
ResponsePlan (PDA, SEARCH, TEXT, {},
              ReturnMessagesSortedByFolder) = True ;
ResponsePlan (PDA, SEARCH, TEXT, {},
              ReturnMessagesSortedBySender) = True ;
ResponsePlan (PDA, SEARCH, TEXT, {},
              ReturnMessagesSortedByPriority) = True ;

-- If there are more than 1 message in the result
-- show a list of the message headers
ResponsePlan (PDA, SHOW, SPEECH,
              {"Wayne Wobcke", 60, "Inbox", NORMAL},
```

B.2 Alkemy Training Examples

```
        ("Paul Compton", 8, "Research", NORMAL)},
ReturnMessageList) = True ;

ResponsePlan (PDA, SHOW, TEXT,
              {"Wayne Wobcke", 60, "Inbox", NORMAL},
              {"Paul Compton", 8, "Research", NORMAL}),
ReturnMessageList) = True ;

-- If user asked to summarize then show the message's summary
ResponsePlan (PDA, SUMMARIZE, SPEECH,
              {"Wayne Wobcke", 60, "Inbox", NORMAL}),
ReturnMessageSummary) = True ;

ResponsePlan (PDA, SUMMARIZE, TEXT,
              {"Wayne Wobcke", 60, "Inbox", NORMAL}),
ReturnMessageSummary) = True ;

-- Collected training examples
ResponsePlan (PDA, SEARCH, SPEECH,
              {"Anh Nguyen", 74, "Inbox", NORMAL}),
ReturnMessageContent) = False ;

ResponsePlan (PDA, SEARCH, SPEECH,
              {"Anh Nguyen", 74, "Inbox", NORMAL}),
ReturnMessageSummary) = True ;

ResponsePlan (PDA, SEARCH, SPEECH,
              {"Wayne Wobcke", 60, "Inbox", NORMAL}),
ReturnMessageContent) = False ;

ResponsePlan (PDA, SEARCH, SPEECH,
              {"Wayne Wobcke", 60, "Inbox", NORMAL}),
ReturnResponse) = False ;

ResponsePlan (PDA, SEARCH, SPEECH,
              {"Wayne Wobcke", 60, "Inbox", NORMAL}),
ReturnMessageSummary) = True ;

ResponsePlan (PDA, SEARCH, SPEECH,
              {"Paul Compton", 24, "Inbox", HIGH}),
ReturnMessageContent) = True ;

ResponsePlan (PDA, SEARCH, SPEECH,
              {"Paul Compton", 24, "Inbox", HIGH}),
ReturnMessageSummary) = False ;

ResponsePlan (PDA, FIND, SPEECH,
              {"John Lloyd",77, "CRC", HIGH},
              {"John Lloyd",32, "CRC", HIGH}), ReturnMessageList) = True ;
```

```
ResponsePlan (PDA, FIND, SPEECH,
  {"John Lloyd",77, "CRC", HIGH}, {"John Lloyd",32, "CRC", HIGH}),
ReturnMessagesSortedBySender) = False ;

ResponsePlan (PDA, FIND, SPEECH,
  {"John Lloyd",77, "CRC", HIGH}, {"John Lloyd",32, "CRC", HIGH}),
ReturnMessagesSortedByPriority) = False ;

ResponsePlan (PDA, FIND, SPEECH,
  {"John Lloyd",77, "CRC", HIGH}, {"John Lloyd",32, "CRC", HIGH}),
ReturnMessagesSortedByFolder) = False ;

ResponsePlan (PDA, FIND, SPEECH, {"John Lloyd",77, "CRC", HIGH},
  {"John Lloyd",32, "CRC", HIGH}), ReturnResponse) = False ;

ResponsePlan (PDA, SHOW, SPEECH, {"John Lloyd",77, "CRC", HIGH}),
ReturnMessageContent) = False ;

ResponsePlan (PDA, SHOW, SPEECH, {"John Lloyd",77, "CRC", HIGH}),
ReturnMessageSummary) = True ;

ResponsePlan (PDA, SHOW, SPEECH, {"John Lloyd",77, "CRC", HIGH}),
ReturnResponse) = False ;

ResponsePlan (PDA, SHOW, SPEECH,
  {"Wayne Wobcke", 60, "Inbox", NORMAL),
  {"Paul Compton", 8, "Inbox", NORMAL),
  {"Charles Gray", 12, "Inbox", LOW),
  {"Amir Michail", 10, "Inbox", NORMAL),
  {"Harvey Tuch", 15, "Inbox", NORMAL),
  {"Ken Robinson", 6, "Inbox", HIGH),
  {"John Davis", 18, "Inbox", HIGH),
  {"Ralf Huuck", 34, "Inbox", HIGH),
  {"Kate Scully", 8, "Inbox", HIGH),
  {"Patrick Caldon", 28, "Inbox", HIGH),
  {"Janet Song", 38, "Inbox", NORMAL}}, ReturnMessageList) = True ;

ResponsePlan (PDA, SHOW, SPEECH,
  {"Wayne Wobcke", 60, "Inbox", NORMAL),
  {"Paul Compton", 8, "Inbox", NORMAL),
  {"Charles Gray", 12, "Inbox", LOW),
  {"Amir Michail", 10, "Inbox", NORMAL),
  {"Harvey Tuch", 15, "Inbox", NORMAL),
  {"Ken Robinson", 6, "Inbox", HIGH),
  {"John Davis", 18, "Inbox", HIGH),
  {"Ralf Huuck", 34, "Inbox", HIGH),
  {"Kate Scully", 8, "Inbox", HIGH),
  {"Patrick Caldon", 28, "Inbox", HIGH),
  {"Janet Song", 38, "Inbox", NORMAL}}, ReturnMessageList) = True ;
```

EVALUATION

C.1 Training Tasks

1. Find all e-mail from Norman. Move it to the 'Friends' folder.
2. Show me my appointments for tomorrow.
3. Do I have a meeting today at 4 pm? Delete it.
4. Schedule a meeting for Friday. Move it to next Monday.
5. Do I have any appointments about conference paper.
6. Do I have any messages from Mark today? Make an appointment with him.

C.2 User Questionnaire

1. It was easy to learn how to use the SPA.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

2. It was easy for me to navigate through the SPA.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

3. Feedback from the SPA was clear and easy to understand.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

4. The SPA worked the way I expected it to.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

Could you please briefly tell us in what ways the SPA did or didn't work the way you expected it to?

5. I felt the SPA understood what I asked it to do.

1	2	3	4	5	6
none of the time	a little of the time	some of the time	a lot of the time	most of the time	all of the time

6. It took a lot of effort to remember how to perform tasks.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

7. It was easy for me to ask questions and make requests in a way the SPA could understand.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

8. The SPA gave me reasonable responses to my questions and requests.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

9. The SPA gave appropriate responses when it didn't understand what I said.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

10. It was easy to recover from any mistakes I made while using the SPA.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

11. Using the SPA is frustrating.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

If you agreed with the previous statement, could you please tell us in what ways the SPA was frustrating to use?

12. The SPA responded to my questions and requests in a timely manner.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

13. The layout of the screens in the e-mail program were clear and easy to understand.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

14. It was easy to find the e-mails I wanted.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

15. It was easy to delete e-mails.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

16. It was easy to move e-mails.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

17. It was easy to move between my e-mail and calendar in the SPA.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

18. The layout of the screens in the calendar program were clear and easy to understand.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

19. It was easy to make appointments.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

20. It was easy to delete appointments in the calendar.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

21. It was easy to change appointments in the calendar.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

22. I was happy about the overall performance of the SPA.

1	2	3	4	5
very unhappy	unhappy	neither happy nor unhappy	happy	very agree

Could you please tell us why you were or were not happy with the overall performance of the SPA?

23. I would use a system like the SPA in future.

1	2	3	4	5
strongly disagree	disagree	neither agree nor disagree	agree	strongly agree

Do you have any other comments to make on what you liked about the SPA, or how we could improve the system? If so please feel free to include them here.

Question	User	1	2	3	4	5	6	7	8	9	10	Mean
<i>It was easy to learn how to use the SPA</i>		5	4	5	4	4	4	4	4	4	4	4.2
<i>It was easy to navigate through the SPA</i>		4	4	4	4	4	3	4	3	4	4	3.8
<i>Feedback from the SPA is clear and easy to understand</i>		4	3	4	4	5	4	5	4	5	5	4.3
<i>The SPA worked the way I expected it to</i>		3	4	5	3	4	4	4	2	4	4	3.7
<i>The SPA understood what I asked it to do a lot of the time</i>		4	4	4	4	4	4	4	3	4	5	4.0
<i>It took a lot of effort to remember how to perform tasks</i>		2	2	2	2	2	2	1	2	2	1	1.8
<i>It was easy to ask questions and make requests that the SPA could understand</i>		4	4	3	4	3	3	5	3	4	4	3.7
<i>The SPA gave reasonable responses to my questions and requests</i>		4	4	5	4	4	4	4	4	5	3	4.1
<i>The SPA gave appropriate responses when it didn't understand what I said</i>		4	4	5	5	2	3	5	3	4	5	4.0
<i>It was easy to recover from mistakes that I made</i>		2	3	4	4	5	3	5	2	4	3	3.5
<i>Using the SPA is frustrating</i>		3	2	4	3	2	2	2	4	4	3	2.9
<i>The SPA responded in a timely manner</i>		4	4	4	4	4	4	4	5	4	4	4.1
<i>The e-mail interface is clear and easy to understand</i>		5	4	4	5	5	4	5	4	2	5	4.3
<i>It was easy to find e-mail messages</i>		4	3	4	5	5	3	5	4	4	5	4.2
<i>It was easy to delete e-mail messages</i>		4	4	4	4	5	2	5	4	5	5	4.2
<i>It was easy to move e-mail messages</i>		5	4	4	5	2	4	4	4	4	5	4.1
<i>It was easy to move between e-mail and calendar in the SPA</i>		4	4	5	4	5	4	5	4	5	5	4.5
<i>The calendar interface is clear and easy to understand</i>		5	3	5	4	5	4	5	4	5	5	4.5
<i>It was easy to make appointments</i>		4	3	4	3	3	3	4	4	4	5	3.7
<i>It was easy to delete appointments</i>		4	4	5	5	4	4	4	4	4	5	4.3
<i>It was easy to change appointments</i>		3	4	4	3	3	4	4	4	4	2	3.5
<i>I was happy about the overall performance of the SPA</i>		4	5	4	4	4	4	5	3	4	4	4.1
<i>I would use a system like the SPA in future</i>		5	4	5	4	3	4	3	4	4	2	3.8

Table C.1: User Survey Ratings

C.3 Utterance-Level Performance for Each User

Subject 1	Number of Utterances	Source of Problem				
		User	Speech	Parser	Dialogue	Back-end
<i>Total</i>	49					
<i>Utterances with All Concepts Recognized</i>	35 (71.43%)					
<i>Unexpected Responses</i>	20		6	3	11 (1 context, 4 object determination, 1 reference, 1 attribute extraction, 3 dialogue strategies, 1 clarification)	
<i>Inappropriate Responses</i>	10			3	7 (1 context, 4 object determination, 1 reference, 1 attribute extraction)	

(a) Subject 1

Table C.2: Performance at Utterance Level

C.3 Utterance-Level Performance for Each User

Subject 2	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	62					
<i>Utterances With All Concepts Recognized</i>	41 (66.13%)					
<i>Unexpected Responses</i>	22	3	16		3 (1 reference, 1 object determination, 1 temporal)	
<i>Inappropriate Responses</i>	4			1	3 (1 reference, 1 object determination, 1 temporal)	

(b) Subject 2

Subject 3	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	39					
<i>Utterances With All Concepts Recognized</i>	26 (66.67%)					
<i>Unexpected Responses</i>	9		8		1 attribute extraction	
<i>Inappropriate Responses</i>	3			1	1 attribute extraction	1

(c) Subject 3

Table C.2: Performance at Utterance Level

C.3 Utterance-Level Performance for Each User

Subject 4	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	48					
<i>Utterances With All Concepts Recognized</i>	37 (77.08%)					
<i>Unexpected Responses</i>	13		6	4	3 (1 semantic analysis, 2 task recognition)	
<i>Inappropriate Responses</i>	7			4	3 (1 semantic analysis, 2 task recognition)	

(d) Subject 4

Subject 5	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	49					
<i>Utterances With All Concepts Recognized</i>	31 (63.27%)					
<i>Unexpected Responses</i>	16	1	12		3 (1 reference, 1 temporal, 1 object determination)	
<i>Inappropriate Responses</i>	3				3 (1 reference, 1 temporal, 1 object determination)	

(e) Subject 5

Table C.2: Performance at Utterance Level

C.3 Utterance-Level Performance for Each User

Subject 6	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	53					
<i>Utterances With All Concepts Recognized</i>	39 (73.58%)					
<i>Unexpected Responses</i>	20		10	3	7 (2 object determination, 2 temporal, 1 attribute extraction, 1 task recognition, 1 dialogue strategies)	
<i>Inappropriate Responses</i>	10			4	6 (2 object determination, 2 temporal, 1 attribute extraction, 1 task recognition)	

(f) Subject 6

Table C.2: Performance at Utterance Level

C.3 Utterance-Level Performance for Each User

Subject 7	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	39					
<i>Utterances With All Concepts Recognized</i>	32 (82.05%)					
<i>Unexpected Responses</i>	9	1	2		6 (4 object determination, 1 semantic analysis, 1 attribute extraction)	
<i>Inappropriate Responses</i>	7			1	6 (4 object determination, 1 semantic analysis, 1 attribute extraction)	

(g) Subject 7

Table C.2: Performance at Utterance Level

C.3 Utterance-Level Performance for Each User

Subject 8	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	107					
<i>Utterances With All Concepts Recognized</i>	62 (57.94%)					
<i>Unexpected Responses</i>	42		26	8	8 (1 context, 3 task recognition, 2 attribute extraction, 1 reference, 1 semantic analysis)	
<i>Inappropriate Responses</i>	25			9	16 (1 context, 3 task recognition, 9 attribute extraction, 2 semantic analysis, 1 temporal)	

(h) Subject 8

Table C.2: Performance at Utterance Level

C.3 Utterance-Level Performance for Each User

Subject 9	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	73					
<i>Utterances With All Concepts Recognized</i>	43 (58.90%)					
<i>Unexpected Responses</i>	30	1	21	5	3 (1 temporal, 2 attribute extraction)	
<i>Inappropriate Responses</i>	11			5	6 (1 temporal, 4 attribute extraction, 1 object determination)	

(i) Subject 9

Subject 10	<i>Number of Utterances</i>	<i>Source of Problem</i>				
		<i>User</i>	<i>Speech</i>	<i>Parser</i>	<i>Dialogue</i>	<i>Back-end</i>
<i>Total</i>	50					
<i>Utterances With All Concepts Recognized</i>	28 (56.00%)					
<i>Unexpected Responses</i>	21	1	15	2	3 (1 object determination, 1 reference, 1 task recognition)	
<i>Inappropriate Responses</i>	7			3	4 (2 object determination, 1 reference, 1 task recognition)	

(j) Subject 10

Table C.2: Performance at Utterance Level

C.3 Utterance-Level Performance for Each User

User		1	2	3	4	5	6	7	8	9	10	Total Per Task
Task												
1	<i>E-mail Search</i>		3	1							3	7
2	<i>Appointment Search</i>											0
3	<i>E-mail Search</i>			1		4				2		7
4	<i>Folder Search</i>		2		1	1				1		5
5	<i>Appointment Schedule</i>								1	2	3	6
6	<i>Appointment Search, Reschedule</i>	4		1		1	3	3	9	2		23
7	<i>Appointment Search, Deletion</i>	6					4		5		1	16
8	<i>Complex E-mail Search</i>	2	10							3	1	16
9	<i>Appointment Search</i>											0
10	<i>E-mail Search, Deletion</i>				1				4			5
11	<i>E-mail Search, Archive</i>	1	3				1	1	2			8
12	<i>E-mail and Calendar Combination</i>	7			2	9		4				22
Total Per User		20	18	3	4	15	8	8	21	10	8	115
Number of Responses		49	62	39	48	49	53	39	107	73	50	569

Table C.3: Number of Clarification Requests by the Dialogue Manager