Introducing a Multi-Dimensional User Model to Tailor Natural Language Generation

by

Jennifer Chu

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 1991

Abstract

Previous work has shown that it is important to include user modeling in question answering systems in order to tailor the output. In this thesis, we develop a natural language response generation model that handles both *definitional* and *procedural* questions, and employs a multi-dimensional user model. The various attributes of the user recorded in the user model include the user's *role*, *domain knowledge*, *preferences, interests*, *receptivity* and *memory capability*. We also address how to represent the user's view of a knowledge base to then tailor generation to that user's view, and introduce a representation for knowledge bases with property inheritance based on the Telos system.

We further specify how this multi-dimensional user model influences different stages of McKeown style generation, on *determining question type*, *deciding relevant knowledge*, *selecting schema*, *instantiating predicates*, and *selecting predicates*. An algorithm is proposed to describe how the generation process can be tailored according to these influences, and some examples are presented to show that the output is desirable.

We also address the problem of *conflicts* arising from the variation in response generation suggested by different user model attributes and use various weighting schemes to resolve them. Furthermore, we include a procedure for updating the user model after each interaction which also enables the algorithm to be used over multiple sessions, with up-to-date information about the users.

Acknowledgements

First I would like to thank my supervisor, Robin Cohen, for suggesting this topic and for her guidance throughout my thesis work. The effort she spent on our discussions, on reading (and re-reading) my thesis, and for suggesting a well-respected university for continuing my studies has been greatly helpful.

I am also indebted to my second readers, Fahiem Bacchus and Paul van Arragon, for their useful comments and pointers to related literature. Chrysanne DiMarco also deserves my thanks for leading our group meetings, and especially for commenting on my writing style.

Thanks also go to Keith Mah for spending the time on solving my LaTeXproblems, helping me with my English, and proofreading my thesis, to Marzena Makuta for being a good friend, who always has time to discuss my work and to have fun, to Jun Shen for being understanding and encouraging, and to my officemate, Paulina Chin, for her delicious desserts and the joy we had together.

Finally, special thanks go to my parents and my best friends, Yi-Jean and Ching-Fang, for their encouragement and emotional support, though from far away.

To my parents...

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 What is Natural Language Generation? 3

1.3 What is User Modeling? 6

1.4 Integrating Natural Language Generation and User Modeling 7

1.4.1 Limitations of Current Systems 7

1.4.2 Our Model 8

1.5 Overview of the Thesis 9

2 Generating Natural Language Using ATNs **12**

2.1 General Problems in Natural Language Generation 12

2.2 McKeown’s Approach 14

2.2.1 System Overview 14

2.2.2 Using Schemata 15

2.2.3 Implementing Schemata with ATNs 17

3 Developing a User Model **20**

3.1 Contents of the User Model 21

3.1.1 Goals and Plans 21

3.1.2 Preferences and Attitudes 23

3.1.3 Knowledge and Beliefs 24

3.1.4 Memory Constraints 26

3.1.5 Interests 27

3.1.6 Receptivity 28

3.2 Dimensions of the User Model 29

3.2.1 Degree of Specialization 29

3.2.2 Modifiability 31

3.2.3 Temporal Extent 32

3.2.4 Method of Use 32

3.2.5 Number of Agents 32

3.2.6 Number of Models 33

4 Motivation for Our Design Decisions **34**

4.1 McKeown’s Approach 34

4.2 Responses with User Modeling 40

5 User Models With McKeown-Style Generation **46**

5.1 The Domain 46

5.2 Knowledge Representation 48

5.3 The User Model 55

5.4 The Interaction Between User Models and Generation 59

5.4.1 Determine Question Type 59

5.4.2 Determine Relevancy 61

5.4.3 Select Strategy 65

5.4.4 Select Proposition 65

5.4.4.1 Predicate Instantiation 66

5.4.4.2 Predicate Selection 67

5.5 Updating the User Model 68

5.5.1 What Should Be Updated? 68

5.5.2 How to Update Them? 69

6 The Algorithm 72

6.1 An Overview of the Algorithm 72

6.2 Determine Question Type 74

6.3 Determine Relevant Knowledge Pool 75

6.4 Select Schema 78

6.4.1 Schemata Available 79

6.4.2 The Procedure 83

6.5 Instantiate Predicate 86

6.6 Select Predicate 96

6.7 Update the User's Knowledge 104

7 Examples **107**

8 Related Work **123**

8.1 Paris’s TAILOR System 123

8.1.1 System Overview 123

8.1.2 The User Model 124

8.1.3 Comparisons 127

8.2 Sarner’s Model 129

8.2.1 The Strategy 130

8.2.2 The User Model 132

8.2.3 Comparisons 132

8.3 Moore’s PEA System 134

8.3.1 System Overview 134

8.3.2 The User Model 136

8.3.3 Comparisons 138

8.4 Wolz’s GENIE System 139

8.4.1 System Overview 140

8.4.2 The User Model 141

8.4.3 Comparisons 142

8.5 McCoy’s ROMPER System 144

8.5.1 System Overview 144

8.5.2 The User Model 147

8.5.3 Comparisons 148

8.6 Sarantinos and Johnson's EXPLAIN system 150

8.6.1 System Overview 150

8.6.2 The User Model 152

8.6.3 Comparisons 154

8.7 Hovy's PAULINE System 155

8.7.1 System Overview 156

8.7.2 Comparisons 158

9 Conclusions **163**

9.1 Summary and Contributions of the Thesis 163

9.1.1 Summary of Our Work 163

9.1.2 Analysis of Our User Model 165

9.1.3 Contributions 166

9.2 Limitations and Future Directions 169

A The Schemata **173**

B Tracing the Algorithm in Detail **175**

Bibliography **184**

List of Tables

3.1 Recall Percentage After Certain Time Intervals 27

9.1 Dimensions of Attributes in Our User Model 166

List of Figures

2.1 TEXT System Overview 16

2.2 The Identification Schema Used in TEXT 17

2.3 ATN for Identification Schema 18

2.4 Ordering of Preference Constraints 19

3.1 Dimensions of User Models 30

4.1 Relevant Knowledge Pool for the Question *What is a Ship?* 36

4.2 Mapping of Schemata to Question Types in TEXT 37

4.3 The Instantiated Predicates 38

4.4 Answer to *What is a Ship* Generated by TEXT 39

4.5 User Model for User A 41

4.6 Desired Output for User A 42

4.7 User Model for User B 43

4.8 Desired Output for User B 43

4.9 User Model for User C 45

4.10 Desired Output for User C 45

5.1 The Cooking Database 49

5.2 Four Entity Types 51

5.3 Representing User's Knowledge in a User Model 58

5.4 Relevant Knowledge Pool for the Question *What is Meringue Pie?* . . 62

5.5 Relevant Knowledge Pool for the Question *How to Make Chocolate Pie?* 63

5.6 Valued Relevant Knowledge Pool 64

5.7 Updated Valued Relevant Knowledge Pool 71

6.1 ATN for the identification schema 80

6.2 ATN for the constituency schema 81

6.3 ATN for the contrastive schema 82

6.4 ATN for the procedural schema 83

6.5 The Pre-determined Schemata for Both Question Types 84

6.6 User's Original Knowledge Base 105

6.7 User's Updated Knowledge Base 106

7.1 User Model in Example 1 108

7.2 Relevant Knowledge Pool in Example 1 109

7.3 Output for Example 1 111

7.4 User Model in Example 2 112

7.5 Output for Example 2 112

7.6 User Model in Example 3 113

7.7 Output for Example 3 114

7.8 User Model in Example 4 116

7.9 Output for Example 4 117

7.10 User Model in Example 5 118

7.11 Output for Example 5 119

7.12 User Model in Example 6 120

7.13 Relevant Knowledge Pool in Example 6 121

7.14 Output for Example 6 122

8.1 The TAILOR System Architecture 125

8.2 Describing a microphone to an expert user 125

8.3 Describing a microphone to a naive user 126

8.4 Definition of Travelers Checks 1 131

8.5 Definition of Travelers Checks 2 131

8.6 The PEA System Architecture 135

8.7 Sample Dialogue with PEA 137

8.8 Response Generated by GENIE 1 141

8.9 Response Generated by GENIE 2 141

8.10 The ROMPER System Architecture 145

8.11 Sample Output of ROMPER 1 146

8.12 Sample Output of ROMPER 2 146

8.13 The EXPLAIN System Architecture 151

8.14 Sample Explanation Generated by EXPLAIN 153

8.15 The PAULINE System Architecture 157

8.16 Sample Output of PAULINE 1 159

8.17 Sample Output of PAULINE 2 160

B.1 Updated User's Knowledge Base 183

Chapter 1

Introduction

1.1 Motivation

Answering a question properly is not an easy task. We start off with an example, given in [Prince, 1981], of two recipes to show how different responses should be provided to different users with different domain knowledge. The two recipes are both for *roast suckling pig*, but with dramatically different details. The first recipe, taken from *Le Repertoire de la cuisine* is used by professional French chefs. It says,

Cochon de Lait Anglaise:

Farcir farce a l'anglaise. Rôtir.

(Translation: English Suckling Pig: Stuff with English stuffing. Roast.)

The second recipe is from a standard American cookbook *The Joy of Cooking*. Part of its description is shown in the following:

Roast Suckling Pig: 10 servings

Preheat oven to 450°.

Dress, by drawing, scraping and cleaning:

A suckling pig
Remove eyeballs and lower the lids. The dressed pig should weigh about 12 pounds.
Fill it with:
Onion Dress, page 457, or
Forcemeat, page 458
It takes $2\frac{1}{2}$ quarts of dressing to stuff a pig of this size. Multiply all your ingredients, but not the seasonings. Use these sparingly until the dressing is combined, then taste it and add what is lacking. Sew up the pig.

.

.

.

Prince claimed that the reason for such a difference between these two recipes is that "the writer of a recipe has a certain set of assumptions about what the reader knows about ingredients, processes, and equipment" ([Prince, 1981]). We believe that in a question answering system, the responses should also be tailored to the users, not based on the system's *assumptions*, but, more precisely, on *what the system knows about the users*.

Now, let's consider a third user who is a restaurant owner. The user is interested in the dish *English suckling pig* not because he/she wants to learn how to prepare it, but because he/she is trying to decide whether it should be included in the menu or not. Obviously, this user has a completely different goal from the chefs, and should expect to get a different response to a question like *what is English suckling pig*. In this case, the following response will be more appropriate.

> The *English suckling pig* is a roasted pig with English stuffing. It is a very fancy dish which takes at least 5 hours to prepare and serves about 10 people.

We see that the responses should depend not only on the user's domain knowledge, but also on the user's role, which implies the user's possible goal (why the user asks the question).

Thus, we believe that, in a question answering system, it is often important for responses to be tailored to specific users. They should be presented at a level that the user can understand and should take into consideration preferences in style which the user may have. The amount of information provided should depend on how willing the user is to receive information and how capable he/she is to remember the details given. Amongst the various characteristics that might influence the *content* and *presentation* of the response, most of the existing natural language generation systems, disappointingly, either do not consider them at all, or cover only part of them.

In this thesis, we will propose an algorithm for combining natural language generation and user modeling which generates natural language responses tailored to specific users.

1.2 What is Natural Language Generation?

Comprehension and *generation* are the two complementary aspects of natural language processing ([Joshi, 1987]), but the latter did not receive much attention until very recently. Literally, natural language generation is the process of generating natural language texts by computers in a written form. It is not simply making computers print out fluent texts, since this can be easily done by using stored texts or template-filling techniques. Instead, "it looks ahead to the time when machines will think complex thoughts and need to communicate them to their human users in a natural way" ([McDonald, 1988]), which means that there is some planning and reasoning involved in the process.

In early work on natural language generation, the main goal is to provide responses to questions in a question answering setting. The process is simply accomplished by using *stored text*, texts which are pre-stored in the system by the system designer. This approach requires the designer to enumerate all the possible questions that might arise and handcode all the answers to them. This causes problems when unpredicted questions arise, and is also inflexible. Later work involves filling in the slots of the templates provided by the generator, which provides more varieties of texts. Templates are instantiated and/or concatenated to provide different outputs, but the system never knows *why* the templates are chosen. Another shortcoming is that the instantiation and juxtaposition of templates sometimes results in awkward or ungrammatical texts. These two methods are considered useful in applications where short response-time is required, and limited generation capability is sufficient.

More recent work includes involving *text planning* strategies in the generation process, having a separate *realization* component to convert the idea to natural language text, etc. Nowadays, we are able to generate not only paragraph-length texts, but also texts with different styles. The systems no longer blindly search for words to fill in the slots, but actually *reason* and *plan* for achieving a certain goal so that they know *how* the output is obtained and *why* they want to say it.

The reasoning and planning ability broadens the applications of natural language generation systems considerably. Current research in natural language generation proceeds not only in developing systems that generate high-quality texts, but also in integrating the generation component with other reasoning modules to fulfill more complicated tasks.

Some work has been done in developing *pure tactical components* which focus more on selecting syntactic structures rather than selecting content. Given content in a particular representation, these systems generate the corresponding English text. McDonald's MUMBLE system ([McDonald, 1986]) and the NIGEL ([Mann, 1983a]) system, which is the realization component of the PENMAN project at ISI ([Mann, 1983b]), both emphasize the production of large varieties of texts that broadly cover English syntax. In this thesis, we focus on the *strategic component* in the generation process (deciding the *content*), rather than the tactical component.

Most current research concentrates on applications of natural language generation. Appelt ([1985]) designed an utterance-planning system, KAMP, that satisfies multiple goals in an utterance to accomplish a certain task. McKeown ([1985]) developed the TEXT system that provides answers in natural language for questions about database structures (this system will be further explained in chapter 2). Paris ([1987]) extended McKeown's work and investigated how a user's domain knowledge might affect an answer. Her system, TAILOR, is a question answering system that makes use of *user models* to provide different *kind* and *amount* of information according to the user's level of expertise. Hovy ([1988a]) studied how *pragmatic* concerns influence the generated text. His system, PAULINE, generates texts describing events under different pragmatic constraints (such as, to increase the hearer's knowledge, to make the tone informal, etc.). This is the first attempt to incorporate *style* into texts using a natural language generation system. Moore ([1989]) looked at the dialogues between advice-givers and advice-seekers. Her PEA system provides a *reactive* approach to explanation from expert systems, one that can accept feedback from users, answer follow-up questions taking previous explanations into account. Sarner and Carberry ([1990]) proposed a strategy for providing definitions in *task-oriented* dialogues. Her method takes into account the user's domain knowledge, possible plans and goals, and the user's receptivity to help decide the final utterance. Sarantinos and Johnson ([1990]) developed a theory of explanation dialogues known as *Extended Schema based Theory (EST)*, which provides more powerful and complete question understanding and explanation generation. Our work will be contrasted with most of the related research later in chapter 8.

1.3 What is User Modeling?

On the most general level, user models are regarded as forming some kind of *mental model* of the user to provide sufficient information for the system to infer the user's beliefs and goals and provide predictive and explanatory power for understanding the interaction of that user with the system ([Wahlster and Kobsa, 1989]).

Recently, user modeling has attracted much research interest in the field of natural language processing, mostly in dialogue or question answering systems (see Computational Linguistics, special issue on user modeling [Computational Linguistics, 1988]). Since these systems have a considerable amount of interaction with the users, it is important to have the users' *images* in the system, just as we do when we talk to people, to help understand the users' utterances and to help provide the most appropriate answers to them. As for what characteristics of the users should be recorded in the user models to provide such an assistance, a detailed discussion will be given later in chapter 3.

There has been some work done on including user modeling in systems for various applications. Rich ([1979]) designed a system, GRUNDY, which acts as a librarian to suggest books to the users. It starts off by classifying the users into several stereotypes, each with some default characteristics, until more specific knowledge about the users is known. The UC system ([Wilensky *et al.*, 1987]), developed at UC Berkeley, is a natural language interface designed to help naive users learn about the UNIX operating system. Four user categories (stereotypes) are used as reference points for inference, and explicitly encoded information about a specific user can be added to override inheritance from the user's category. McCoy ([1986]) looked into the problem of the possible *misconceptions* a user might have when interacting with the system. She developed a system, ROMPER, that reasons about the possible sources of misconceptions on an individual basis, and generates cooperative responses based on this reasoning. Cohen *et al*. ([1989]) proposed a model for providing responses specific to a user's *goals* and *background*. Their system, ThUMS, is implemented in a domain of diagnosing a student's learning disabilities and models both the student and the user (who seeks a diagnosis for the student).

1.4 Integrating Natural Language Generation and User Modeling

1.4.1 Limitations of Current Systems

Of all the generation systems mentioned in the previous two sections, some contain no user modeling at all, therefore, providing unique output to the same input under all circumstances ([McKeown, 1985], [McDonald, 1986], [Mann, 1983b]). Some of them realize the importance of tailoring responses to users, but take information about the user as input rather than having an explicit user model, or obtain it from previous discourse ([Moore, 1989], [McCoy, 1985], [Hovy, 1988a]). In the former case, there is the problem of annoying a user by repeatedly asking questions at the beginning of every interaction, while in the latter case, the system might not get sufficient information.

We have mentioned that *stereotyping* is often used for user modeling ([Rich, 1979], [Wilensky *et al.*, 1987]). Since it is often difficult to strictly classify to which class a user should belong, this method may be impractical. Some systems concentrate on only one or two of the user's aspects that are considered relevant to the generation process, such as the user's knowledge, or the user's possible plans and goals ([Paris, 1987], [Moore, 1989], [Sarner and Carberry, 1990]). In fact, there are many more characteristics of the user that will influence the response from the system, and an algorithm for natural language generation becomes much more complicated when taking all these characteristics into account (as will be shown later in the thesis).

Some of the existing systems emphasize the *tactical component* either without a reasoning process or with a limited reasoning process for choosing the content ([McDonald, 1986], [Mann, 1983b], [Hovy, 1988a]). Thus, these systems have to be combined with some other reasoning mechanism that can serve as the *strategic component* to make a complete natural language generation system.

Updating the user model is also an important issue that has not yet been discussed sufficiently. Paris ([1987]) did sketch a solution to the problem, but in her case the updated information is kept only within the same conversation, without being recorded permanently in the user model. Thus, in the next interaction, the system possesses out-of-date information about the user.

1.4.2 Our Model

Due to the aforementioned limitations of existing systems, we propose a model of a response generation system that includes a multi-dimensional user model, and discuss the interactions between the user model and the generation system.

Our model is based on the TEXT generation system developed by McKeown ([1985], to be discussed in chapter 2). It extends TEXT to handle both *definitional* and *procedural* questions, and includes a multi-dimensional user model to capture information about the users. In addition to the user's knowledge and user's goals[1] that have been discussed quite often in the past, we introduce the user's *preferences*, *interests*, *receptivity*, and *memory constraints* to further help make decisions in the generation process. Also, a Telos-like representation (for Telos, see [Koubarakis *et al.*, 1989]) for recording the user's view of a knowledge base has been developed. Providing these aspects in the user model, we study how they influence different stages in the generation process, namely, *determining*

[1]This aspect is captured by the attribute *user's role* in our user model.

question type, *deciding relevant knowledge*, *selecting schema*, *instantiating predicate*, and *selecting predicate*.

From the study of the multi-dimensional user model and its interaction with the generation process, we obtain knowledge of both *what* should be recorded in a user model and *when and how* these aspects influence the generation process. Based on this knowledge, we propose an algorithm specifying how generation can be tailored to different users according to the information provided in the user model. We also hand-trace the algorithm on various examples in a cooking domain to show that it indeed generates satisfactory responses.

1.5 Overview of the Thesis

The remaining chapters of this thesis have contents as follows. In chapter 2, we discuss the general problems that arise in natural language generation and introduce McKeown's TEXT system, upon which our model is based. The basic techniques that our model inherits from TEXT will also be discussed.

Chapter 3 discusses how to develop a user model, emphasizing the information that should be recorded. This is based on the general discussion of user models presented in Kass and Finin [1988], also adding some attributes that we propose. For each attribute, we give some examples or justification to argue that it indeed should influence the content of the responses.

The motivation that leads us to our proposed model for natural language generation with user modeling is described in chapter 4. We first show the output given by TEXT in the domain of naval intelligence, for the question *what is a ship*, followed by three examples with outputs tailored to individual users asking the same question. We argue that these outputs are appropriate for each corresponding user model, and use them as the goal to aim for — to develop a response generation model that generates high-quality tailored responses like them.

Chapter 5 presents most of the main points in this thesis. We introduce a new application domain of cooking, which admits definitional and procedural questions, and discuss its characteristics. The contents proposed for the user model are introduced and a representation for the user's domain knowledge is presented. We also discuss how the user model and the generation process interact at various stages, and illustrate our method of updating the user model after every conversation.

An algorithm systematically describing the interaction discussed in chapter 5 is presented in chapter 6. We give an outline of the algorithm in the first section, and a more detailed description of each step in the following sections.

Some examples in our cooking domain, hand-traced through the algorithm, are shown in chapter 7. In this chapter we present examples of different users and show how the responses generated by the algorithm differ for each user. The important decision points for each example are shown and an explanation is given for matching the actual output and the aspects given in the corresponding user model. A more detailed trace of the algorithm for the first example is given in appendix B, for interested readers.

Chapter 8 discusses related work. A number of natural language generation systems are briefly introduced and compared to our model. In summary, our model is different from other existing systems in that we present a general framework to show how various aspects of a user may tailor responses in a question-answering system, and develop an algorithm describing when and how these influences take place.

Finally, in chapter 9, we give a summary of the work done in this thesis and of our contributions, then discuss the limitations of our model and possible directions for future research.

Chapter 2

Generating Natural Language Using ATNs

As described in chapter 1, there are a number of natural language generation system-s developed for various purposes, such as providing explanations in an expert system environment and providing responses to database queries. In this chapter, I discuss the general problems encountered in natural language generation and introduce McKeown's ATN-based approach, upon which my model is based.

2.1 General Problems in Natural Language Generation

One major difference between natural language generation and natural language understanding is that the former requires more *decision-making* processes while the latter involves more *disambiguating* processes. The problems that arise in generation include: ([Hovy, 1988a],[Mann, 1987],[McKeown, 1986])

1. *What to say.* This is concerned with deciding the content to be generated, while avoiding saying what is too obvious or too complex, and yet providing sufficient information to make the generated text comprehensible.

2. *How to say it.* This concerns the way of expressing the content, how to organize sentences and how to choose words to make the text coherent. The problems include:

   (a) Ordering of text

   (b) Clarifying what to say by adding supporting text

   (c) Determining sentence boundaries

   (d) Deciding when to use anaphora (noun phrases that refer to objects mentioned earlier in the sentence or in a previous sentence, [Allen, 1987])

   (e) Choosing lexicons

3. *Why it should be said.* This includes the consideration of the *style* (also called *pragmatics* in [Hovy, 1988a]) of the generated text, whether text should be enhanced or emphasized in a particular fashion for the audience, etc.

Generally speaking, a text generation system can be divided into two stages, the *strategic component*, which determines the content and structure of discourse, and the *tactical component*, which uses a grammar and dictionary to realize in the target language the message produced by the strategic component ([McKeown, 1985]). The strategic component determines *what to say* and part of *how to say it* (e.g., (a) and (b) above, the parts of 2 which relate to content and structure), while the tactical component determines the rest of *how to say it*. As for *why it should be said*, Hovy ([1988a]) argues that rhetorical goals and strategies, pertaining to stylistic and opinion-based considerations, are reasons for a generator to choose one output over another. These rhetorical goals and strategies interact with both the strategic component and the tactical component, and therefore influence both what to say and how to say it.

Our proposed model emphasizes the influences of user models on the content and structure of the generated text, i.e., how they influence the decisions made in the strategic component. Therefore, in the following section, I will introduce McKeown's TEXT system ([McKeown, 1985]), a system which focuses on the strategic component.

2.2 McKeown's Approach

McKeown's TEXT system is designed to generate paragraph-length responses to simple questions about information in a database. It uses text planning strategies to organize text, and a focus mechanism to help make decisions.

2.2.1 System Overview

TEXT is developed within the framework of a natural language interface to a database system that addresses the specific problem of generating answers to questions about the database structure. Three types of questions are allowed, namely, *definition*, *information*, and *comparison*, each having predefined schemata (patterns of discourse structure, for details, see section 2.2.2) acting as templates to help generate answers.

The TEXT system consists of four decision-making components — the semantic processor, the schema selector, the schema filler, and the tactical component. In addition to these basic modules, it also includes a database[1], which provides the necessary infor-

[1]We use the term database here interchangeably with knowledge base, to refer to the knowledge base representation of the underlying database.

mation for answering questions, and a set of schemata for constructing responses. The control flow of TEXT is shown in figure 2.1. Questions are answered by first partitioning off a subset of the knowledge base (which represents the database in question) determined to be relevant to the given question into the *relevant knowledge pool* (**Determine Relevancy**). Then, based on the question type, a schema encoding partially ordered *rhetorical predicates* is chosen (**Select Strategy**). These predicates, together with a *focus mechanism*, determine the selection of propositions from the relevant knowledge pool, which constitutes the content and order of the answer (**Select Propositions**). This message is then passed to the tactical component which transforms the message into English.

2.2.2 Using Schemata

TEXT uses discourse strategies, called *schemata*, for selecting information from the underlying knowledge base and ordering it.

McKeown examined a number of expository texts, noted four different patterns and represented them as schemata, namely, *identification*, *attributive*, *constituency*, and *contrastive*. The identification schema is shown in figure 2.2, where "{}" indicates optional constituents, "/" indicates alternatives, "+" indicates that the item may appear 1 to n times, and "$*$" indicates that the item may appear 0 to n times. The *identification* schema captures a strategy used for providing definitions. Its characteristic techniques are represented as *predicates* which include identification of an item as a member of a generic class (**identification**), description of an object's constituency or attributes (**constituency/attributive**), analogies (**analogy**), and examples (**particular-illustration/ evidence**). The *constituency* schema describes an entity in terms of its subparts. The *attributive* schema is to illustrate a particular point about a concept or an object. The *contrastive* schema is used to describe something by contrasting a major point against a negative

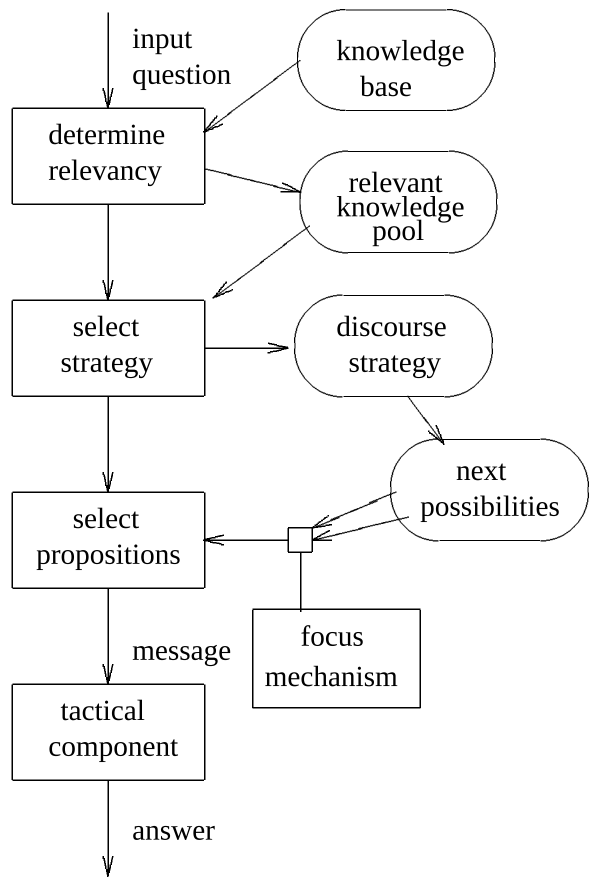


Figure 2.1: TEXT System Overview

Identification Schema

Identification (class & attribute / function)

{Analogy / Constituency / Attributive}*

Particular-illustration / Evidence$^+$

{Amplification / Analogy / Attributive}

Figure 2.2: The Identification Schema Used in TEXT

point. The last three schemata are shown in detail in appendix A.

2.2.3 Implementing Schemata with ATNs

The four schemata are implemented in TEXT using a formalism based on a parsing strategy called an *augmented transition network (ATN)* (turned around to be used for generation) ([Woods, 1970]). The corresponding ATN for the identification schema is shown in figure 2.3. Starting from the state *ID/*, the ATN is traversed by taking an arc at a time, which either proceeds to the next state or remains in the same state. Note that taking an arc corresponds to the selection of a proposition for the answer and the states correspond to filled stages of the schema. The subroutines (e.g., subr description) each consist of a choice of several predicates which are not shown in the diagram. The process continues until a *pop* arc is chosen, which terminates the response.

Most of the states in the ATNs have more than one outgoing arc, which indicates that there is often more than one choice for the next proposition. TEXT solves this problem by introducing a *focus mechanism*, which tracks what is considered to be in focus. The

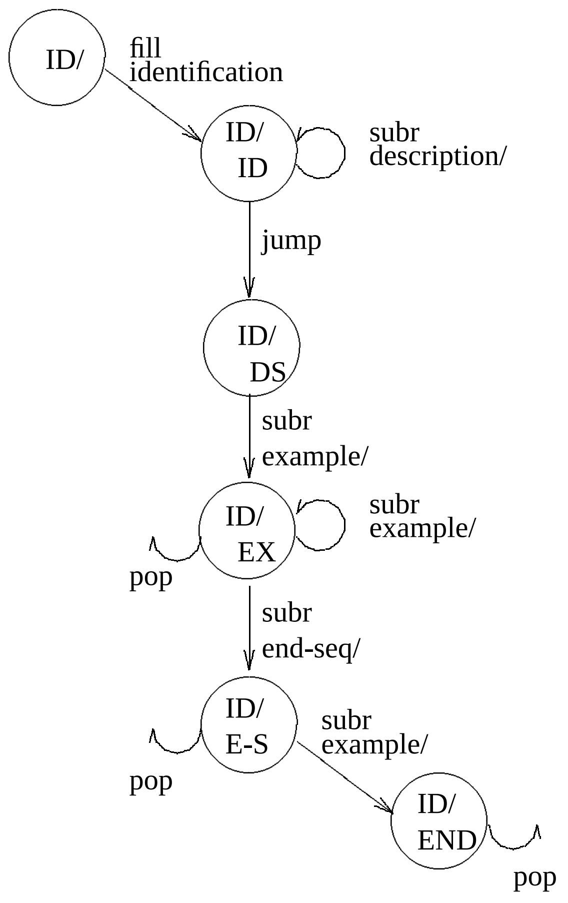


Figure 2.3: ATN for Identification Schema

1. Shift focus to item mentioned in previous proposition
2. Maintain focus
3. Return to topic of previous discussion
4. Select proposition with greater number of implicit links to previous proposition

Figure 2.4: Ordering of Preference Constraints

rules for focusing constrain the focus to be the object being asked, or something closely related to it. Figure 2.4 shows a preference ordering of how the focus can be shifted or maintained. Whenever there is more than one candidate proposition, the one with the highest priority in the preference list is chosen.

A brief example of TEXT responding to the question *what is a ship* will be shown in chapter 4. The example serves to illustrate the relevant knowledge pool (which encodes the part of the knowledge base in focus) and how the preference for focus shifts affects the generation process.

Chapter 3

Developing a User Model

The TEXT system, as described in chapter 2, does not include any user modeling. The consequence is that the system provides the same answer, no matter who asks the question. This is undesirable, since it is obvious that in our daily life we vary the content and structure of conversations depending on our audience. For example, we explain what an IC is to an electrical engineering student and an arts student differently, in terms of the level of detail, the word choices, etc. In order to tailor generated text to the user's point of view, we need user models to record aspects of different users. However, some researchers in the user interface and information retrieval community have reacted against the idea of having a user model. They contend that it is impractical to model a user with all sorts of *ill-formed* aspects of human cognition (e.g., how do we know when a user believes that a specific fact is true?). Therefore, instead of making the system do all the reasoning to satisfy the user's needs, they claim that it should be the *user*'s own effort to make inferences from what the system provides, and that one generic interface will suffice ([McKeown, 1990]). In spite of this controversy, we still believe that involving user modeling in a generation system is profitable. In this thesis, we will show that user modeling indeed contributes to deciding the content of the response.

When dealing with a user model, there are three important issues to be considered, namely, the *representation*, the *content*, and the *usage*. We will discuss all three of these issues when presenting our proposed user model in chapter 5. The representation of a user model mainly deals with the problems of how to record the user's *knowledge and beliefs* and his/her *goals*. Various people have worked on this issue, proposing methods such as *linear parameter representation* ([Rich, 1979]), *meta-language predicates* ([Konolige, 1981]), *possible-world semantics* ([Appelt, 1985]), etc. In chapter 1 we gave a brief discussion of the usage of user models, i.e., how user models can be built into other systems to help generate responses tailored to specific users. In this chapter we will concentrate on discussing the possible content of a user model.

3.1 Contents of the User Model

Kass and Finin ([1988, 1989]) proposed that for a system and a user to interact cooperatively, the information that a system needs to have about a user includes knowledge about the user's goals and plans, preferences and attitudes, and about the user's knowledge and beliefs. In addition, we maintain that the user's interests, memory capabilities and receptivity are also important factors that should influence natural language generation. I will discuss each of these user attributes in turn in this section.

3.1.1 Goals and Plans

A user's possible goals and plans can definitely influence the content of generated text. In a paper by Sarner and Carberry ([1988]), when the question *what is baking soda* is asked, the system infers the user's possible goal using previous discourse and information available in the user model. For the purpose of baking a cake, the answer will be *baking* *soda, when heated, releases carbon dioxide and makes the cake rise*; for the purpose of relieving indigestion, *baking soda, when dissolved in water, produces a chemically basic solution that will counteract acidity* will be appropriate. This demonstrates that knowing a user's plan may be very important in deciding appropriate generation.

Although we realize that knowing the user's plans and goals is important, it is not an easy task. The simplest case is one where the user directly states his/her goal[1]. But unfortunately, in many cases, the speaker expects the hearer to infer the implicit goals[2]. In some cases, the user might have an implicit goal at a higher level than the one explicitly stated[3]. Therefore, much more additional information about the user and previous discourse must be known to infer the most plausible goal.

Some previous work has been done on recognizing a user's plans and goals. The most common method for dealing with plan recognition is to have a *plan library* available in the system and to find the most appropriate plan by inferring from information about the user and past discourse. Perrault, Allen and Cohen ([Perrault *et al.*, 1978]) used some predefined *operators* to recognize *speech acts* and further infer the user's actual intent in a dialogue understanding system. Litman ([1986]) discussed the representation and recognition of domain-independent *discourse plans* and domain-dependent *domain plans* to understand the implicit relationships between utterances.

Carberry ([1983]) dealt with the recognition of plans and goals and used them to formulate appropriate responses. The system includes a set of plans which are hierarchical structures of component goals and actions, and constructs the user's plan as the dialogue

[1]For instance, in the interaction: *I'd like to major in Computer Science. What courses should I take now?*

[2]An example is the *indirect speech act* proposed by Searle in [Searle, 1975]. The question *can you reach the salt?* should be interpreted as a request to perform the action of passing the salt.

[3]The question adapted from [Carberry, 1983], *Is Professor Cohen teaching CS686 in fall term?* may require different answers according to the user's ultimate goal (to take CS686, to take Cohen's course, etc.).

progresses by inferring a lower-level goal and relating it to potential higher-level plans. McKeown, Wish and Matthews ([McKeown *et al.*, 1985]) proposed an approach of integrating *plan recognition*, *user modeling*, and *explanation generation*. The user's goal is derived within a discourse segment, and they focus on how the content of the explanation should vary according to the goal inferred. Van Beek ([1987]) employed a database of domain dependent plans and goals, and a user model, recording various information including the user's immediate discourse goals and higher level domain goals. The plan for achieving the domain goal is inferred and an appropriate response for accomplishing the goal is provided to the user.

3.1.2 Preferences and Attitudes

Paris ([1987]) analyzed the styles of descriptive texts from a variety of sources, ranging from adult encyclopaedias to high school textbooks. She found that people tend to describe objects to experts and naive users differently, and proposed two description strategies for users at different levels of expertise. The *constituency schema*, posited by McKeown ([1985]), which breaks an object into subparts and describes each of them individually, is suitable for experts, while for naive users, a *process trace*, which describes the procedure of how the object works is used more often. For example, in the Collier's Encyclopedia ([Halsey, 1962]), a telephone is described by first stating that it consists of a transmitter and a receiver, then describing each part in turn, whereas the Britannica Junior Encyclopedia ([Britannica Junior Encyclopedia, 1963]) gives, in the first paragraph,

> When one speaks into the transmitter of a modern telephone, these sound waves strike against an aluminum disk or diaphragm and cause it to vibrate back and forth in just the same way the molecules of air are vibrating.

This provides evidence that different *styles* of presentation are appropriate for different people. The fact can be due to the person's level of expertise or, simply, his/her own preference.

A natural language generation system also has to take a user's attitudes into account when generating responses. The user's attitude can be thought of as the user's *point of view*, which might sometimes be biased, i.e., the user might particularly like/dislike a certain object. When the user's attitude is not neutral, the system can choose to be cooperative or uncooperative. In the former case, the system avoids mentioning what the user dislikes, while in the latter, the system tries to correct the user's misconceptions. The IMP system ([Jameson, 1988]), which responds to questions from a user concerned about evaluating an object, adopts the former approach.

3.1.3 Knowledge and Beliefs

It is obvious that what a user knows should influence the system's responses. For a system to provide helpful information, it should say what is *to the point*. To illustrate this, Sarner and Carberry ([1988]) proposed the *Principle of Usefulness*, following the Grice's Maxims ([Grice, 1975])[4]. It says,

[4] *Grice's maxims* have four categories,

1. *Quantity*. Make the contribution as informative as is required, but no more informative than is required.
2. *Quality*. Do not say what you believe to be false or what you lack adequate evidence.
3. *Relation*. Be relevant, say things at an appropriate stage of the transaction.
4. *Manner*. Be perspicuous, avoid ambiguity, be brief and orderly.

1. The response should be made at a high enough level that it is meaningful to the user.
   - (a) Do not say something the user will not understand.
   - (b) Do not give information that addresses more detailed aspects of the user's task-related plan than is appropriate for his current focus of attention.
2. The response should be made at a low enough level that it is helpful to the user.
   - (a) Do not inform the user of something he already knows.
   - (b) Do not give information that is unrelated to the user's goals and task-related plan, or is too general for his current focus of attention in the plan.

To serve this purpose, the system must know the user's knowledge and beliefs. I use the terms as follows: *knowledge* refers to the *long-term* knowledge that the user believes to be true, while *beliefs* are *short-term*, applicable to the current conversation. Knowledge can be further classified into *domain knowledge*, which is specific for the application domain, and *world knowledge*, also called *commonsense knowledge*, helping the system to interact with the user in a more appropriate manner ([Kass and Finin, 1988]).

A fair amount of work has been done in investigating how the user's knowledge affects the system's responses. Paris ([1987]) argued that the user's knowledge influences not only the *amount* of information provided, but also the *style* of presentation. In fact, in addition to influencing the generation process at the *content* level, it also affects the *tactical component* of the generator. This comes in later work by Bateman and Paris ([1989]) on tailoring the *syntactic phrasing* to specific users. Cohen and Jones ([1989]) incorporated user modeling into an expert system for educational diagnosis, in which responses are limited by the user's *domain knowledge*, and the *user's preference*[5]. In their later work ([Cohen *et al.*, 1989]), they included the user's *goals* to further augment the responses, and examined the influence of the interaction between the user's knowledge and goals.

3.1.4 Memory Constraints

Some natural language generation systems maintain the new information that the user learns for the duration of the conversation only ([Paris, 1987]); therefore, the user is assumed to *have forgotten* what he/she has learned when interacting with the system the next time. Other systems assume the user to be an *ideal learner*, one who remembers everything forever ([Moore, 1989]). Unfortunately, neither of these views of a user is entirely accurate.

Johnson performed an experiment to determine how well people recall material they have read after certain time periods ([Johnson, 1970]). He had a group of students read an article, and asked them to reproduce as much of the article as they could after retention intervals of 15 minutes, 7 days, 21 days, or 63 days. The average recall percentage for each of the time intervals is shown in table 3.1. Though the percentage of recall depends on the structural importance of the article, we see that even after 15 minutes, only slightly more than half of the material was recalled correctly. This percentage decreases as the time interval increases, to about 16% after 63 days. This supports our claim that a memory constraint is indeed an important factor in natural language generation, since without it, the system will often assume the user knows what he/she has actually forgotten.

Seaman ([1980]) mentions that people recall better if *cue words* are used. MacLeod

[5]Here the attribute *preference* is used differently from ours. Cohen defined preference as *what the user is more concerned with*, which is similar to our *interest* attribute.

time interval	15 min.	7 days	21 days	63 days
correct percentage of recall	60.0	32.3	25.5	15.8

Table 3.1: Recall Percentage After Certain Time Intervals

([1988]) also argues that people are able to re-learn what they have forgotten better than learning something completely new. Therefore, we propose that reminder is necessary, but that the actual content can be mentioned more briefly than the first time it was introduced.

In order to capture the user's memory capability, we can classify users into several stereotypes, each having memory capability ranging from *low* to *high*. Each class has a default value indicating the approximate percentage of information that will be forgotten after a certain period of time so that a different amount of reminders can be provided to different classes of users.

3.1.5 Interests

Another factor that should be considered is the user's interests. Corbett ([1990]) suggests that we can make a subject more attractive to an audience by showing that it is important, momentous, curious or *relevant to the interest of the audience*.

Interests can be stated either explicitly or implicitly. The latter suggests that a user's interest can be inferred from other knowledge of the user. For example, a person with a good deal of knowledge about the Toronto Blue Jays or the Montreal Expos will be considered as being interested in baseball (inference from user's knowledge), while one who asks about recipes often might be thought of as being interested in cooking (inference from user's goals). Nonetheless, interests can also be inferred from the user's background, sex, age, etc.

To show how the user's interests actually tailor natural language generation, suppose the question *who is Benjamin Franklin* is asked. For a general answer, the system may respond *Benjamin Franklin, an 18-century American, was a scientist, politician, and writer*. However, for further information, we want to be more user-specific, by *guessing* what the user might want to know. For a physics student, the answer will be *he flew a kite in the thunderstorm and proved that lightning is electrical in nature*. For a political science student, we might say that *he was a member of the Constitutional Convention of 1787, and had a hand in the writing of the Declaration of Independence of America*. In a third case, an arts student might prefer *he had a number of publications, including a series of 14 essays signed "Silence Dogood", as well as a weekly newspaper called "the New-England Courant" in 1921*. The above, however, are obtained by inferring the user's interests from their background.

As mentioned earlier, Cohen *et al*. ([1989]) take into account the user's *preference*, which is similar to the *interest* attribute described here. They pre-define certain preferences for each class of users and provide responses according to them.

3.1.6 Receptivity

Hovy included in his rhetorical goals of style, the *openmindedness* of the hearer, indicating his/her willingness to consider new topics ([Hovy, 1988a]). Our definition of *receptivity* is similar to Hovy's openmindedness, but also considers the user's *ability* to absorb the information given, in order to determine how much to say.

A person with high receptivity is assumed to be eager to learn new things and to be able to understand the newly introduced material well. Therefore, we might not only want to say what he/she asks for, but also add more relevant information to support what we say.

To illustrate how receptivity influences natural language generation, let us take the question *what is a peach* for example. If it is asked by a person with low receptivity, what we might say is *peach is a kind of large fruit, usually round, with downy white or yellow skin flushed with red, highly flavoured sweet pulp and a rough stone*; for a person with high receptivity, we can say more about a peach, or even introduce nectarines, and compare the two fruits.

A user's receptivity can be recorded in the user model in a way that is similar to the one used for recording the user's memory constraints. Several classes ranging from low to high indicate how willing the user is to accept new concepts, which helps decide the amount of information to be included in the responses.

3.2 Dimensions of the User Model

There are many ways of designing a user model. Kass and Finin ([1988]) analyzed six dimensions of a user model, namely, *degree of specialization, modifiability, temporal extent*, *method of use*, *number of agents* and *number of models*, as shown in figure 3.1. In designing a user model, one has to make choices for each of these dimensions. These choices are based on the application domain, the number of users, and the kinds of attributes included in the user model, etc. In chapter 9 we review the user model presented in this thesis and classify it according to the dimensions listed in figure 3.1.

3.2.1 Degree of Specialization

The two extreme cases of specialization are *generic* and *individual*. The generic case is basically making the same assumptions about *all* users, without actually tailoring re-

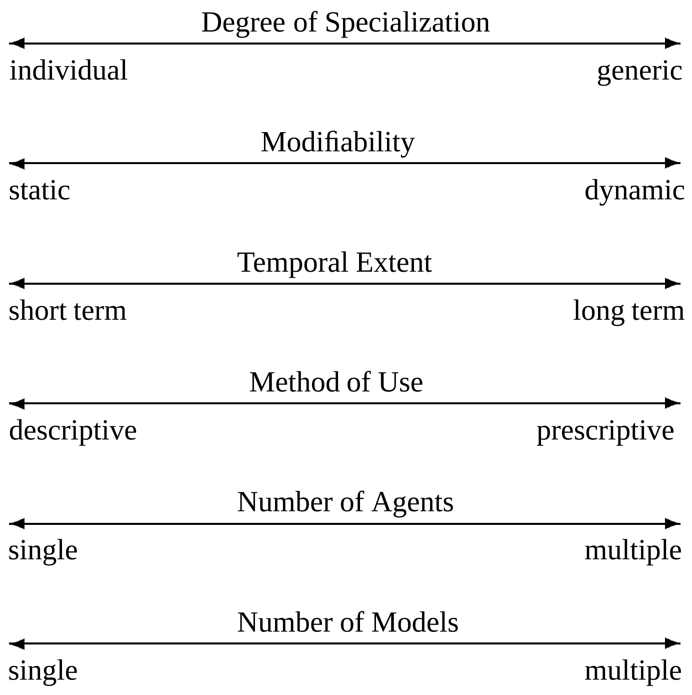


Figure 3.1: Dimensions of User Models

sponses to individual users, while the individual case is treating every user differently, which will become very costly if the system has a large number of users.

A compromise of these two cases is to categorize the users into several *stereotypes*, with some basic assumptions associated with each type of user. A user can be classified into one of these roughly defined stereotypes, with some additional information indicating where the facts about the user and the default values of the type disagree. This is the model most widely adopted in existing user modeling systems.

The degrees of specialization for all of the contents mentioned in the previous section do not have to agree. It is best to choose the degree of specialization according to the characteristics of the contents, so as to serve the needs of both efficiency and accuracy. For instance, in a course-advising system, we can assume the generic goal for all users to be *to get a degree*, whereas the database of what courses the user has taken can be of several stereotypes, categorized according to the year and department the user is in. On the other hand, preferences and receptivity tend to be more specific to individuals.

3.2.2 Modifiability

A user model can be *static* or *dynamic*. A static user model remains the same no matter what new information about the user is learned, while a dynamic user model is updated after each interaction or a certain period of time.

The discussion in section 3.1.4 suggests that a user model that does not update its knowledge of the user does not make much sense. Therefore, for a system to provide the most appropriate response to the user, its user model has to be up-to-date, i.e., *dynamic*. However, part of the user model can be static, e.g., sex, some information might change gradually over time, e.g., interests, and other information should be updated frequently, e.g., knowledge.

3.2.3 Temporal Extent

The temporal extent of a user model can be either *short term* or *long term*. Short term information refers to information that is kept only during the same conversation, but not recorded in the user model after the interaction. This can include the user's mood, the conversational environment, and so on, which tend to change for every interaction. A user model which is *generic* and *dynamic* is usually short-term. Since information about a particular user will not be kept for long, the user model can be updated quite often. Long term information, on the other hand, usually implies that the information is kept in a *static* or *individual* model.

3.2.4 Method of Use

A user model can be used *descriptively* or *prescriptively*. Most of the current systems adopt the descriptive use of a user model, which means acting *passively* by maintaining a database about the user, looking up and updating the database whenever necessary.

A prescriptive use of the user model can be thought of as acting *actively*. Instead of using the database as something to check up, the system uses this knowledge to *anticipate* the user's reaction. Jameson's IMP system ([Jameson, 1988]), for example, uses an *anticipation feedback loop* to predict the user's reaction to the system's response.

3.2.5 Number of Agents

In most cases, the system deals with one agent at a time, providing information to the person it is interacting with. The system keeps information on this user, from its own point of view. But sometimes, the *main role* in the conversation is not the user him/herself. The user might be a lawyer who is consulting the system to find out the best way to protect his/her client, or a mother who is consulting a medical expert system to find out what is wrong with her child. In these cases, the situation gets much more complicated. The system must keep a model for the client (in the first case) as well as the lawyer so that it can tailor the responses to the lawyer's *point of view* of the client.

3.2.6 Number of Models

People often play multiple roles, depending on where they are, to whom they are talking, etc. If the information for these roles is kept in the same model, conflicts might arise. For example, a man, asking his subordinate to give him a report on the design of a new product, might expect the report to be formal and to the point. But when asking his son how he did at school today, he will expect the atmosphere to be more colloquial. This shows a conflict in the person's *preference of style*.

One way of solving this problem is to keep multiple models of the same agent, choosing one for the conversation depending on the possible role he/she is playing, the conversational atmosphere, and so on. In more complicated situations, the system will have to decide when to switch models even during the same conversation.

Chapter 4

Motivation for Our Design Decisions

In order to see how user modeling influences natural language generation, we take a closer look at some examples in McKeown's TEXT system domain of naval intelligence. We choose the typical *what is a ship* question, look at how the system generates the response, then specify a few user models and create some *desired responses* for each of them. By comparing the desired output and TEXT's original output, we come up with an algorithm that generates natural language text tailored to individual users which will be presented in chapter 6. In this chapter, we first give an example describing how generation is done in TEXT, then present three users with different characteristics asking the same question, together with the desired responses. This sheds some light on what we expect to achieve in our work.

4.1 McKeown's Approach

We gave a brief introduction of how the TEXT system works in chapter 2. In this section, we will give an example of how TEXT generates the answer to the question *what is a ship*.

TEXT operates in a database containing information about vehicles and destructive devices. When the question *what is a ship* is asked, the first step is to construct the relevant knowledge pool. The relevant knowledge pool in figure 4.1 is constructed by selecting the area in the database immediately surrounding the ship (for details on how this is done, see [McKeown, 1982]).

Next, the schema selection process is performed. In TEXT, the selection of discourse strategy (schema) is based on a pre-defined mapping which assigns different schemata to different question types, as shown in figure 4.2. Some of the question types have more than one candidate schema. The final decision is made by consulting the relevant knowledge pool for the amount of available information. For example, the identification schema and the constituency schema are the two candidates for questions requesting definitions. If the relevant knowledge pool contains more information about the object in focus than about the object's children, the identification schema is chosen; otherwise, the constituency schema will be used. This is usually done by choosing a pre-determined level in the database hierarchy at which most of the objects appearing above that level have more information about their children, and the ones appearing below the level have more information about themselves.

In this example, the *what* question falls into the category of definitional questions. Since the ship occurs below the pre-determined level in the hierarchy, the identification schema is chosen.

After selecting the schema, TEXT begins traversing the ATN graph corresponding to the schema chosen. The ATN graph for the identification schema was shown earlier in figure 2.3. The first arc dictates that the response must begin with a proposition matching the identification predicate. Searching through the relevant knowledge pool, the predicate is instantiated and translated to *a ship is a water going vehicle that travels on the surface*.

Now, we are in the ID/ID state in the ATN. For the next sentence, we can either choose

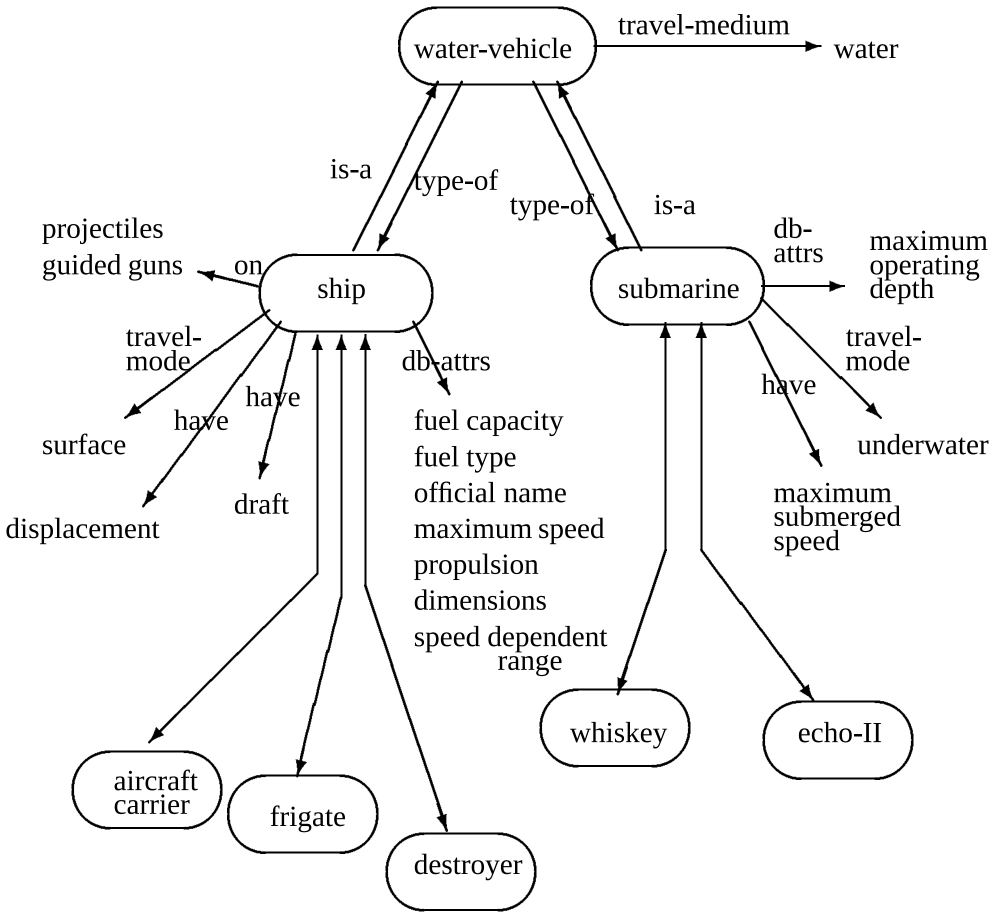


Figure 4.1: Relevant Knowledge Pool for the Question *What is a Ship?*

Requests for definitions
- Identification
- Constituency

Requests for available information
- Attributive
- Constituency

Requests about the differences between objects
- Contrastive

Figure 4.2: Mapping of Schemata to Question Types in TEXT

the arc *subroutine description*, which leads us back to the same state, or choose subroutine *example* and move on to the next state. Since the subroutine *description* includes three choices, *attributive*, *constituency*, and *analogy*, and the subroutine *example* consists of *particular-illustration* and *evidence*, we have five candidate predicates in total.

To choose among these candidates, TEXT first matches the predicates in the relevant knowledge pool to instantiate them, then applies the focus constraints shown in figure 2.4, if more than one predicate is applicable. The instantiated predicates and their foci are shown in figure 4.3, where the particular-illustration predicate is left out because there is no information available at this point.

Now following the preferential ordering on how focus of attention should shift (figure 2.4), the evidence predicate has the highest priority since it focuses on an element that was just introduced, *the surface-going capabilities*. Therefore, it is chosen as the second proposition.

1. Analogy

   The ship carries guided projectiles and guns.

   focus = ship

2. Constituency

   There are 5 types of ships in the ONR database: aircraft carriers, frigates, ocean escorts, cruisers, and destroyers.

   focus = ship

3. Attributive

   The ship has DB attributes *maximum speed*, *propulsion*, (*fuel capacity* and *fuel type*), *dimensions*, *speed dependent range* and *official name*.

   focus = ship

4. Evidence

   Its surface-going capabilities are provided by the DB attributes *displacement* and *draft*.

   focus = surface-going capabilities

Figure 4.3: The Instantiated Predicates

(definition SHIP)

; What is a ship?

Schema selected : Identification

> 1. A ship is a water-going vehicle that travels on the surface. 2. Its surface-going capabilities are provided by the DB attributes *displacement* and *draft*. 3. Other DB attributes of the ship include *maximum speed*, *propulsion*, *fuel* (*fuel capacity* and *fuel type*), *dimensions*, *speed dependent range* and *official name*. 4. The DOWNES, for example, has maximum speed of 29, propulsion of *stmturgrd*, fuel of 810 (fuel capacity) and *bnkr* (fuel type), dimensions of 25 (draft), 46 (beam), and 438 (length) and speed dependent range of 4200 (economic range) and 2200 (endurance range).

Figure 4.4: Answer to *What is a Ship* Generated by TEXT

TEXT continues traversing the ATN in a similar method until a *pop* arc is chosen, which ends the generation process. The complete response for this question is shown in figure 4.4, where the path taken in the ATN is,

$$\begin{array}{lll} \text{ID/} & \text{– identification} & \longrightarrow \text{ID/ID} \\ & \text{– jump} & \longrightarrow \text{ID/DS} \\ & \text{– example} & \longrightarrow \text{ID/EX} \\ & \text{– end-seq} & \longrightarrow \text{ID/E-S} \\ & \text{– example} & \longrightarrow \text{END} \end{array}$$

4.2 Responses with User Modeling

The procedure and output discussed in the previous section are *standard* in TEXT, i.e., no matter who asks the questions, the system provides this unique answer. We believe that a *user-friendly* system should be able to tailor the responses to different users given the user model of each. In this section, we will look at some examples that show how information about the user might affect the response. Note that the users in the following examples represent real people to demonstrate that the combinations of attributes in their user models are possible.

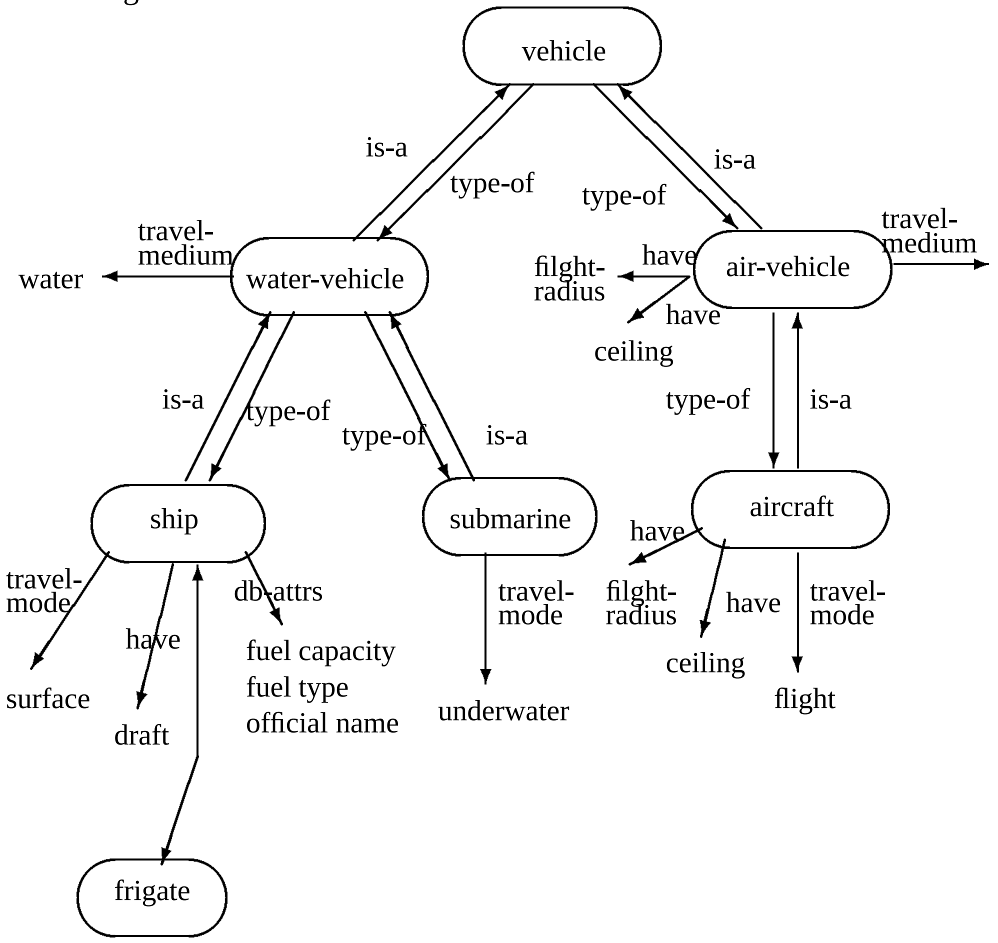
User A:

In this example, we present a user who is a high school student interested in aircraft, with the user model shown in figure 4.5. Being a teenager, it is very likely that he/she has high receptivity and a good memory. Since this user is not an expert in this domain, presenting the information using examples might be more appropriate[1]. He/She knows some aspects about a ship, its sibling *submarine*, and an air vehicle *aircraft*. Now, if the question *what is a ship* is asked by this user, we will expect an answer as shown in figure 4.6.

In the desired response, we see that two schemata, identification and contrastive, are chosen, because the user has both high receptivity and good memory. The former suggests that the user is willing to learn more new information and the latter indicates that the user has a higher possibility of remembering what he/she was told. Sentences one to three are generated by the identification schema, whereas the last two are both examples, which the user prefers. The second proposition gives the example of an aircraft carrier because of the user's interest in aircraft. The fourth sentence is generated by the contrastive schema, choosing the ship's sibling *submarine*, which the user knows about, for the comparison.

[1]This information is all taken as input in our model. Here we are just showing that this hypothetical person makes sense as a possible *real* person.

1. Knowledge:



2. Interest: aircraft

3. Preference: example

4. Receptivity: high

5. Memory capability: high

Figure 4.5: User Model for User A

(definition SHIP)

; What is a ship?

Schema selected : Identification, Compare and Contrast

> 1. A ship is a water-going vehicle that travels on the surface. 2. An aircraft carrier, for example, is a type of ship with a *displacement* between 78000 and 80800 and a *length* between 1039 and 1063. 3. A cruiser is another type of ship that carries between 8 and 42 torpedoes, between 4 and 98 missiles and between 1 and 4 guns. 4. A ship is different from a submarine in that a submarine travels under water and has a DB attribute *maximum operating depth*[2].

Figure 4.6: Desired Output for User A

User B:

In this example, we demonstrate how the user's preference and memory capability influence the response. The user in this example is a aircraft maintenance engineer who has a poor memory. He/She has the same knowledge as the high school student (perhaps more knowledge about aircraft, but since this does not influence the response for our question, we will not show this part of the knowledge base for this user), and most likely is interested in aircraft. Since a maintenance engineer is an expert in a particular type of vehicle, he/she should have high receptivity in this domain; it is also natural that he/she prefers the response being *descriptive* (prefers attributes). The user model is shown in figure 4.7. It differs from user A's model in that the preference is *attributive* and the memory capability is *low*. For user B, the appropriate response to the question *what is a ship* is shown in figure 4.8.

1. Knowledge: same as user A in figure 4.5
2. Interest: aircraft
3. Preference: attributive
4. Receptivity: high
5. Memory capability: low

Figure 4.7: User Model for User B

(definition SHIP)

; What is a ship?

Schema selected : Identification

1. A ship is a water-going vehicle that travels on the surface. 2. Its surface-going capabilities are provided by the DB attributes *displacement* and *draft*. 3. Other DB attributes of the ship include *maximum speed*, *propulsion*, *dimensions*, *speed dependent range*, as well as *fuel capacity*, *fuel type*, and *official name*. 4. Aircraft carriers, frigates, and cruisers, for example, are different types of ships.

Figure 4.8: Desired Output for User B

In this example, only one schema, the identification schema, is chosen, because the user's memory capability is low. Also because of his/her a poor memory, some information that appears in the user's knowledge base shows up again in the response, for instance, the attribute *draft*, and some DB attributes *fuel capacity*, *fuel type*, and *official name*. Compared with the answer given to user A, this response comprises much more attributes, but fewer details in the examples. This is due to the user's preference in *attributive*, rather than in *examples*.

User C:
This particular user model shows how the influences of memory capability and receptivity take place. The person presented here is an old general with the user model shown in figure 4.9. This general is also interested in aircraft, and because of his age, it is understandable that he has low receptivity and poor memory. He happens to have the same domain knowledge as the previous two users and also prefers attributive information. The only difference between the general and the previous engineer is his low receptivity; therefore, we expect the output of the question *what is a ship* to be as shown in figure 4.10.

Obviously, this response contains much less information than the first two. Attributes are mentioned, but not as many as in the answer for user B. The reason is that user C has low receptivity, proposing that he is less willing to learn new information. Also, only one example, the aircraft carrier, is given, for the same reason.

1. Knowledge: same as user A in figure 4.5
2. Interest: aircraft
3. Preference: attributive
4. Receptivity: low
5. Memory capability: low

Figure 4.9: User Model for User C

(definition SHIP)

; What is a ship?

Schema selected : Identification

> 1. A ship is a water-going vehicle that travels on the surface. 2. The DB attributes of a ship include *maximum speed*, *propulsion*, *dimensions*, *speed dependent range*, as well as *fuel capacity, fuel type*, and *official name*. 3. An aircraft carrier, for example, is a type of ship.

Figure 4.10: Desired Output for User C

Chapter 5

User Models With McKeown-Style Generation

We introduced natural language generation and user modeling individually in previous chapters, but the most important issue is *how they interact*, i.e., *when* and *how* the user model influences the decisions made by the generator to provide different responses to different users. In this chapter, we introduce a user model for recording the information about the user mentioned in section 3.1, and discuss when and how this information influences the generation process.

5.1 The Domain

The original ONR database used by McKeown contains information about vehicles and destructive devices ([McKeown, 1985]). The questions that can be asked include *definition*, *information*, and *comparison*, the answers to which all provide *static* information.

Maybury ([1990]) suggests that there are four generic prose forms, *description*, *nar-*

ration, *exposition*, and *argument*. Description includes definition, comparison, analogy, etc., and is used for describing people, places, and things. Narration is for writing about events, and can be reports, stories, or biographies. Exposition conveys ideas and methods, for example, instructions, processes, and propositions, while argument contains deduction, induction, and persuasion.

The types of questions that McKeown deals with all fall into the category of description, which is only a small portion of possible text forms. In order to broaden the scope, the model we propose uses a *cooking domain*, in which information about various dishes as well as the recipes for certain dishes are available. In this domain, in addition to the definitional questions (e.g. *what is X?*), *procedural* questions can also be asked (e.g. *how is X made?*), which provides *dynamic* information about the steps for making X.

Part of the cooking database is shown in figure 5.1. The whole database is represented in a *semantic network*, with the most general concept, *food*, as the root. The objects in the domain are organized in an *ISA* hierarchy, with the descendents inheriting the attributes of their ancestors. There are several kinds of nodes in the tree, *subclasses of food* (classes), *specific dishes* (tokens), *subplans*, and *primitive actions*. The higher level nodes in the tree are more general concepts of food, while most of the nodes in the third level from the bottom are specific dishes, along with their associated recipes. Some recipes can be decomposed into subplans, each subplan being a child of that dish. The leaf nodes, then, are primitive actions used in the parent subplans. Although all nodes in the hierarchical structure inherit attributes from their ancestor nodes, there exists a particular case where a child node has an additional value with the same attribute name as its parent's. In this case, we introduce a notation of an attribute value prefixed by a '+' sign indicating that the same attribute name appears in one of its ancestor nodes as well. For example, *meringue pie* is a subclass of food, inheriting the attributes from its ancestors, *pie*, *pastry*, *dessert*, and *food*. The attribute *parts: + top meringue* shows that in addition to the part *meringue*, there are also other parts recorded in its ancestor nodes ; therefore, we search in the node *pie* and find that the other parts are *crust* and *filling*. Meringue pie has two children, namely, *lemon pie* and *chocolate pie*. To make a lemon pie, the *crust*, *filling*, and *meringue* have to be prepared separately, therefore, each being a subplan of making a lemon pie. Furthermore, for making the pie crust, we have *cut shortening*, *make dough*, and *roll dough* listed as primitive actions.

5.2 Knowledge Representation

The database shown in figure 5.1 is presented at an abstract level. In order to store it in the system, we need a formal representation for it. We use the knowledge representation language Telos ([Koubarakis *et al.*, 1989]) as a guideline to record both the system's and the user's knowledge. Telos adopts an object-oriented representational framework, based on ideas from semantic networks and frame-based representations. It provides a hierarchical structure for representing the parent-child relationships and the inheritance between parents and children.

The main objects in Telos are *classes*, *tokens* and *attributes*. Simple classes are defined as classes with only tokens as instances. Classes other than simple classes are referred to as *meta-classes*. Attributes of classes are classes as well; attributes of tokens which are instances of classes have attributes which are instances of these attribute classes.

Another major feature of the Telos language is the ability to specify temporal information with classes and tokens. We have not employed this feature in the examples used in this thesis, but for future work we can examine the usefulness of recording and tracking the time intervals during which certain objects and relationships are held in the knowledge base.

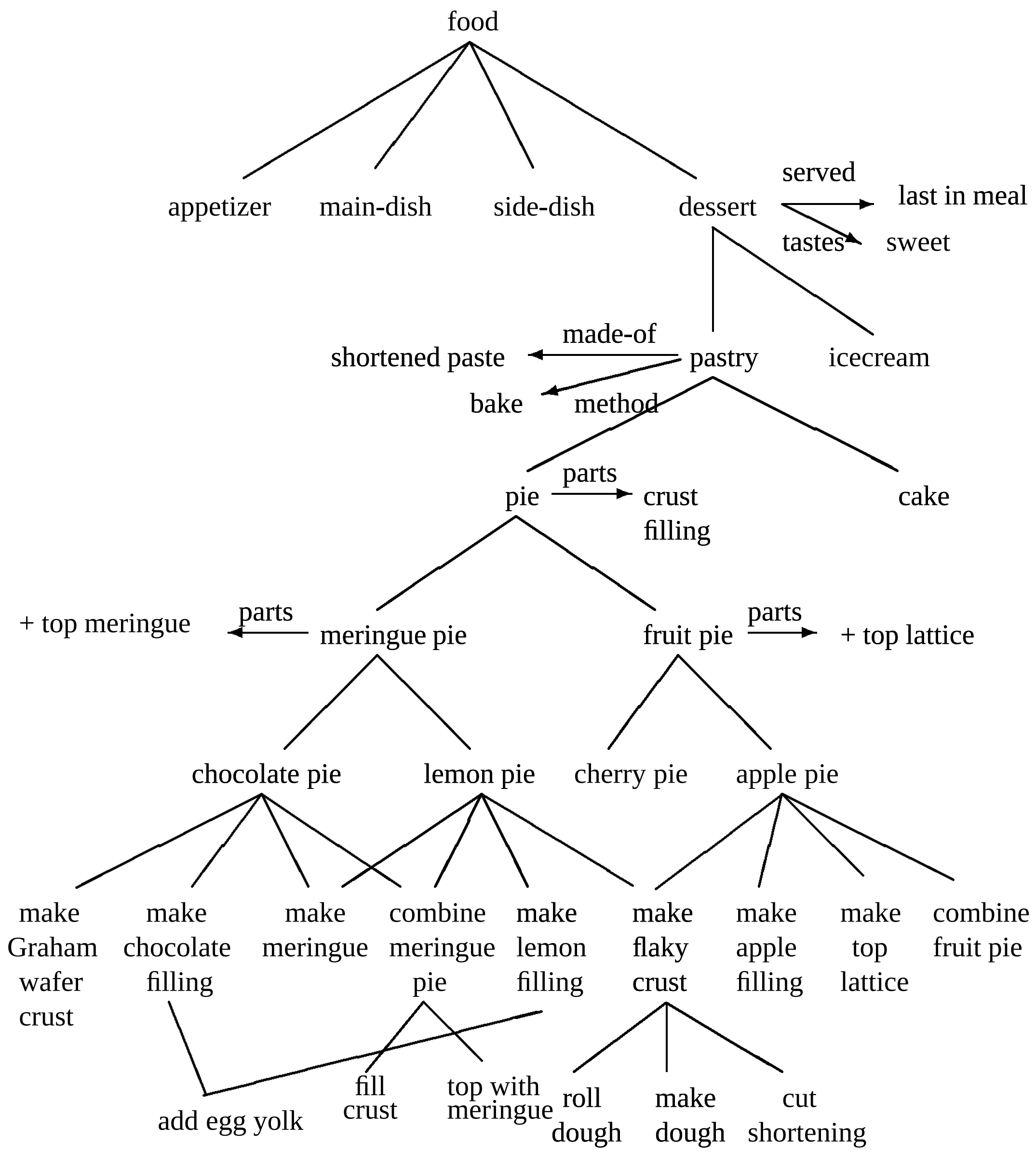


Figure 5.1: The Cooking Database

We use the framework from Telos, but make some modifications to suit our particular needs. We include attribute information and also consider procedural information attached to tokens. There are four entity types for nodes at various levels in the hierarchy, shown in figure 5.2.

The first entity type *class* serves as a more general category. The *ISA* slot indicates the link between the object and its parent. The *WITH* slot includes all the attributes associated with this object, the attribute names together with their values. Within the *CHILDREN* slot are the links between the object and its children. Most of the nodes in the top few levels in the hierarchy will fall in the category *class*. For example, the node *pastry* in figure 5.1 will be represented as,

```
CLASS pastry
    ISA dessert
    WITH
        made-of: shortened paste
        method: bake
    CHILDREN
        pie
        cake
END
```

The second type *token*, is used to indicate objects that cannot be further instantiated. *Subplans* are attached at the token level[1]. For tokens *INSTANCE_OF* slot contains the object's parent in the hierarchy, the most specific class to which it belongs. The *PROCE-*

[1]Note that it would be possible to have a specialization hierarchy for procedural information as well. In this case, a procedure could be attached at a higher class level (e.g. meringue pie), and then instantiated or augmented at the token level (e.g. lemon pie). We did not explore this option.

```
CLASS class_name
   ISA parent_class_name
   WITH
      list of attributes
   CHILDREN
      list of children
END

TOKEN token_name
   INSTANCE_OF parent_class_name
   PROCEDURE
      list of subplans
END

SUBPLAN subplan_name
   INCLUDED_IN
      parent_token_names
   INGREDIENTS
      list of ingredients
   WITH
      primitive actions
   PROCEDURE
      actual procedures
      (including primitive
      actions)
END

PRIMITIVE_ACTIONS action_name
   USED_BY
      parent_subplan_names
   DESCRIPTION
      descriptions
END
```

Figure 5.2: Four Entity Types

DURE slot will include all the links to its children (subplans), if any; otherwise, it will be a description of how to make the dish. The token *lemon pie* will be filled in as,

```
TOKEN lemon pie
    INSTANCE_OF meringue pie
    PROCEDURE
        make crust: make flaky pie crust
        make filling: make lemon filling
        make top: make meringue
        combine: combine meringue pie
END
```

Note that the subplans in the *PROCEDURE* slot are in the format *X:Y*, where *X* is a more general concept, being a collection of several more specific subplans, and *Y* is one of the instantiated subplans within *X*. The purpose of this is to provide information for the contrastive schema. With this representation we know that the *make lemon filling* in lemon pie corresponds to the *make chocolate filling* in chocolate pie, both being in the class *make filling*, so they can be listed as one of the differences between lemon pie and chocolate pie.

The next level, *subplan*, covers the children of *tokens*. These are designed to be a high-level decomposition of the recipes so that some of them can be shared among several tokens. The *INCLUDED_IN* slot gives us the links to its parents, i.e., all the dishes that include this subplan. The *INGREDIENTS* slot provides us with all the basic ingredients used in this particular subplan (in general, this slot can be used to specify objects required in order for the procedure to be carried out). The *WITH* slot gives us the links to the *primitive actions* that are used in this subplan. The last slot, *PROCEDURE*, provides us with the detailed procedure for that part of the recipe. For the *flaky pie crust* subplan, we can denote it as:

```
SUBPLAN flaky pie crust
    INCLUDED_IN
        lemon pie
        apple pie
    INGREDIENTS
        1 cup all-purpose flour
        1/2 tsp salt
        1/3 cup shortening
        3 Tsps very cold water
    WITH
        cut shortening
        make dough
        roll dough
    PROCEDURE
        1. Sift flour and salt into a bowl, cut shortening into flour (cut
        shortening).
        2. Sprinkling water over the flour and mix until it forms a dough
        (make dough).
        3. Roll the dough (roll dough) to a thickness of 0.5cm.
        4. Transfer the dough to pie pan and trim the edge.
        5. Bake at 220C for 10-12 minutes.
END
```

The last entity type *primitive action* is a further decomposition of the subplans, which might appear in several subplans. For instance, the primitive action *make dough* appears not only in making pie crust, but also in making pizza crust, tea biscuits, etc. There are only two slots in this entity, the *USED_BY* and the *DESCRIPTION* slots. The former includes its parents, all the subplans that use this primitive action, while the latter describes how the action is actually done. The primitive actions and subplans are used not only because they allow for efficient storage (by using the same subplan/primitive action for all the recipes they are included in), but also because they control the level of detail provided in the answer to a procedural question (by choosing whether to expand a subplan/ primitive action or not). The primitive action *cut shortening* can be expressed as,

```
PRIMITIVE ACTION cut shortening
  USED_BY
    flaky pie crust
    pizza crust
    tea biscuits
  DESCRIPTION
    Cut shortening into flour with a pastry blender or
    two knives until no pieces of shortening larger than
    a pea appear, when the bowl is shaken.
END
```

To effectively search this knowledge base, we propose an implementation where any node of the knowledge base could be indexed. These indices would be built not only on the system knowledge base, but also on every user's individual knowledge base. They would be used to locate the current focus in the knowledge base and to check if an object in the relevant knowledge pool appears in the user's domain knowledge.

Providing these four entity types and the indices, we will be able to represent and search through the entire system database as well as the user's knowledge in the user model very efficiently.

Note that extending the representation language to include subplans is particularly useful for domains such as cooking, with questions such as *how to make X*. Other extensions to the representation language can be considered, when building natural language generation systems which handle a broader range of question types. See Maybury ([1990]) for a discussion of a wider range of possible question types for natural language generation.

5.3 The User Model

As discussed in chapter 3, a user model contains information about a particular user or a group of users in order to make the system respond more appropriately for individual users. We also suggested what information the user model should provide, though it might differ slightly from domain to domain. In this cooking domain, the information included in the user model is quite similar to that in the ship domain described in chapter 4, with an additional attribute *user's role*. It consists of,

- User's knowledge: this will be an overlay of the system database, indicating the part the user knows about. An example is shown in figure 5.3. The nodes denote the objects known to the user, and an arc indicates that the user knows the relationship between the two nodes (not just the nodes themselves). Note that in the example, the user does not know the concepts *meringue pie* and *fruit pie*; therefore, the user's database is structured differently from the system's, having lemon pie and apple pie as children of pie (we call this *incomplete information*). The user may know part of the attributes of a class (e.g. dessert), and may also know a class without knowing its relationship to other classes (e.g. icecream).

- User's role: This provides clues for inferring the user's possible plans and goals. In our cooking domain, we stereotype the users as either *chefs* or *diners*, with the assumption that a chef would like to know more about how to make the dish, i.e., asking for the recipe, while a diner is more concerned with what the dish consists of, how it tastes, etc.

- User's preferences: This concerns the *style* of responses the user prefers. We give two options here, *attributes* or *examples*. The former focuses on the aspects of the object being asked about, while the latter gives instances of it. For example, for a user who prefers attributes, the answer to *what is pastry* will be *pastry is a kind of dessert which is made of shortened paste, and is often baked*, while for one who prefers examples, the response can be *pastry is a kind of dessert; pies and cakes are instances of pastry*.

- User's interests: The most important usage of user's interests in this domain is to help the system decide which examples to choose. The interests can be particular dishes or flavours that the user is interested in learning. When giving an example of meringue pie, the system will choose lemon pie if the user is interested in lemon, or chocolate pie if he/she is interested in chocolate.

- User's memory capability: The user's memory capability will be indicated as either *high* or *low*, indicating how well the user is expected to remember material introduced, and therefore, determining whether reminders should be added to the responses or not[2].

- User's receptivity: This is also marked as *high* or *low*, and determines the amount of *new* information introduced to the user. This amount varies by the number of

[2]Here we assume that the user interacts with the system through keyboard and terminal only. The case where the user writes down the system's responses to compensate his/her poor memory is not covered.

schemata chosen and the number of predicates chosen within each schema. An example of choosing two schemata was shown in the answer provided to user A in section 4.2.

Conflicts exist in how the information in a user model influences generation. Memory capability and receptivity, for example, both influence the *amount* of information to be included, but sometimes contradict each other. Low memory capability implies that it is quite unlikely that the user will remember what he is told; therefore intuitively, we will expect the system to say less, and expect the user to remember what is said. Although this user has a poor memory, it does not prevent him/her from being eager to learn new things, i.e., having high receptivity. The latter suggests that the system say more in order to satisfy the user's desire. In this case, there is a conflict as to whether more information should be provided or less. Our solution is to concentrate on the *reminders*, i.e., repeating *old* information, as far as memory capability is concerned, but focus on introducing *new* information with regard to receptivity. Therefore, in the previous case, reminders will be provided because of low memory capability, and new information will be introduced because of high receptivity.

One question here is how we obtain this kind of information about the user. There are two possible solutions to this question. The first is to have an *interview session* with a new user to find out all the necessary information. The second is to *stereotype* a new user to a certain class and use the default values of that class to begin, with, then update the parts where the default values disagree with the user's actual characteristics during the conversation. In our model, we assume that the information in the user model is always correct, and the attributes, except user's knowledge, will not be automatically updated (for details of updating the user model, see sections 5.5 and 6.7). Hence, we make the assumption that a new user will be interviewed at the beginning of the conversation in order to initialize the user model. Note that since we do not handle the user's *misconceptions*

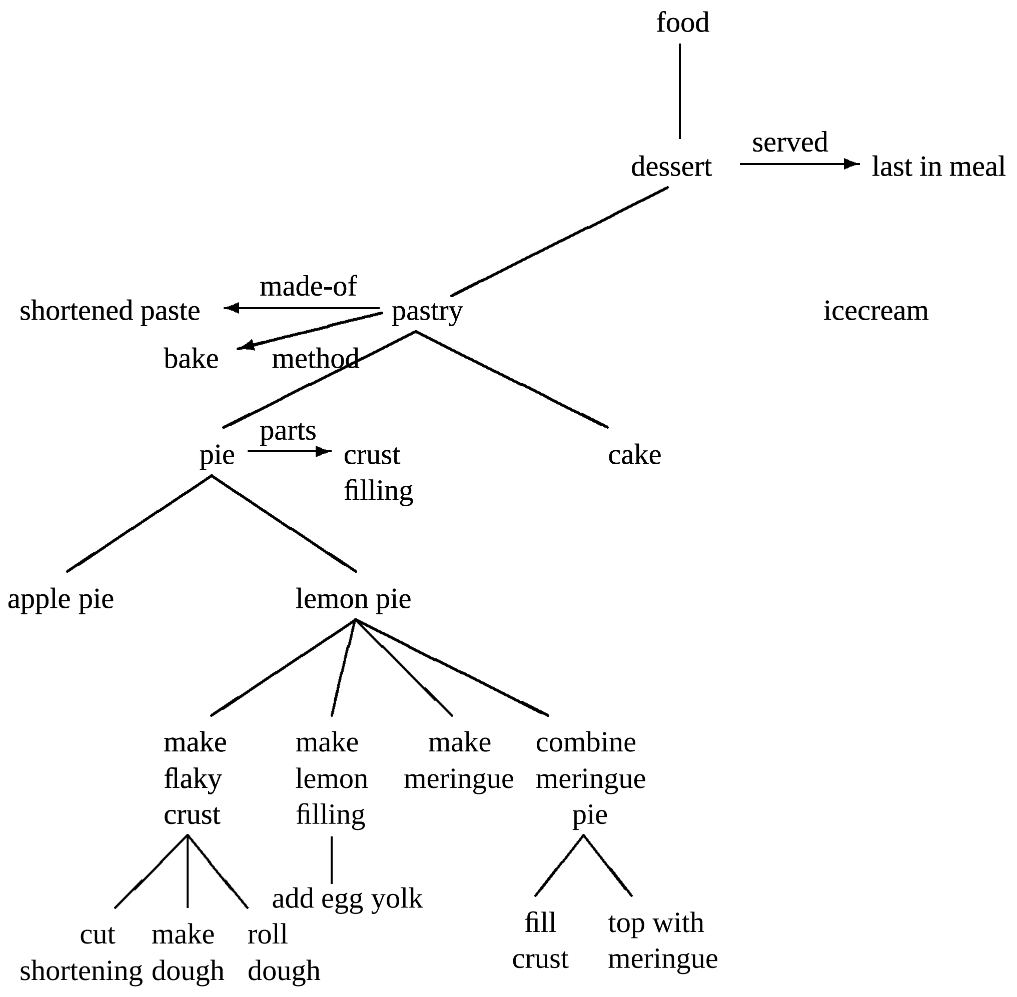


Figure 5.3: Representing User's Knowledge in a User Model

(assuming an incorrect relationship between two objects or associating with an object an attribute it does not have), we have to make sure that the user's knowledge agrees with the system's. This could be realistically achieved by examining the user's knowledge base after the interview session. If a misconception is detected, the incorrect information will not be recorded in the user model (either the *link* or the *attribute* is omitted)[3].

5.4 The Interaction Between User Models and Generation

We briefly introduced McKeown's TEXT system in chapter 2. As shown in figure 2.1, the strategic component of the TEXT system includes *determine relevancy*, *select strategy*, and *select propositions*, which decide *what to say* in the response. Adding user modeling actually influences all three stages of this process. Furthermore, since more than one question type is allowed in our model, the actual type that the user intends has to be decided by consulting the user model. We will explain how these interactions take place briefly in this section, and go into more detail when we discuss the algorithm in chapter 6.

5.4.1 Determine Question Type

We discussed earlier that there are two types of questions that can be asked in our system, *definitional* questions and *procedural* questions. The users can ask questions in either of the two forms *what is X* or *how is X made*, but how does the system decide whether the question asked is intended to be a definitional or a procedural one? We claim that the question type is closely related to the user's possible plans and goals, and can be predicted

[3]Here we act *passively* by omitting the user's misconceptions. A better approach can be to provide information to the user to correct his/her misconceptions.

by the user's role in the user model.

In our domain, we assume that the user's role is either a chef or a diner, where the former is interested in knowing how to prepare a dish and the latter merely wants to know the ingredients, how it is prepared, to decide if he/she will order it or not. Knowing the possible goals for the two types of persons, we can analyze the *what* and *how* questions and decide which categories they fall into when being asked by users of different roles.

The *how is X made* question is quite obvious, because this explicitly states that the user wants to know the procedure for making X. Thus, it is definitely classified as a procedural question[4]. For the *what is X* question, it becomes more complicated. As pointed out by Paris ([1988]), when a *what* question is asked, the answer can either be obtained by traversing a constituency schema, giving all the subparts that constitute the object, or by giving a process-trace, describing how the object works. In our case, we can answer the question by giving the ingredients, its taste, and even some examples, which the diner will prefer, or by providing a step-by-step instruction for making that dish, which the chef will be interested in.

Note that the idea of deciding the question type according to the user's role can be applied to other domains, but the mapping between roles and plans is domain-dependent. Therefore, in another domain, a different mapping of the users' roles and their possible goals should be developed. Once the possible goal is available, we can infer whether definitional or procedural information will be more appropriate.

[4]Of course, the level of detail in a recipe provided to a chef and a diner should be different. Here the user's role only decides the question type since the level of detail can be determined by the user's knowledge which should differ considerably between a chef and a diner.

5.4.2 Determine Relevancy

In McKeown's original model, when a question is asked, part of the knowledge base in focus is selected to form the relevant knowledge pool. All further processing relies on the relevant knowledge pool, and moreover, all information in the pool has equal preference of being chosen.

We agree with McKeown on forming the relevant knowledge pool to provide constraints on what can be said, but we believe that some parts of the relevant knowledge pool should have higher priorities for being said than others. Therefore, our process of creating the relevant knowledge pool can be divided into two stages: first, selecting relevant knowledge; and second, assigning *importance values* to all the nodes and arcs selected.

The first part is done as in TEXT, by concentrating on the global focus to section off relevant information in the knowledge base. In order to do so, the most important thing is to determine the global focus, which is the object being asked about. For a definitional question, we choose the relevant knowledge to be the global focus's descendants to the subplan level, its ancestors to the top[5], its siblings and the siblings' children. With the cooking database shown in figure 5.1, the resulting relevant knowledge pool for the question *what is meringue pie* is shown in figure 5.4. As for a procedural question, where the object being asked about must be a token or subplan, the relevant knowledge pool will be the subtree with the global focus as root, and the parents of all the subplans and primitive actions of the global focus. If the question *how is a chocolate pie made* is asked, the

[5]In the cooking domain, it is more appropriate to choose the ancestors to the second level from the top, since the class *food* is a very general concept in this domain, and it does not make much sense to define an object as a kind of food. But if *food* is further defined as a kind of *physical object*, we still want to cut all the nodes higher than food in the hierarchy. Therefore for future work, we should investigate how to specify *where* to cut off the ancestor chains in various domains.

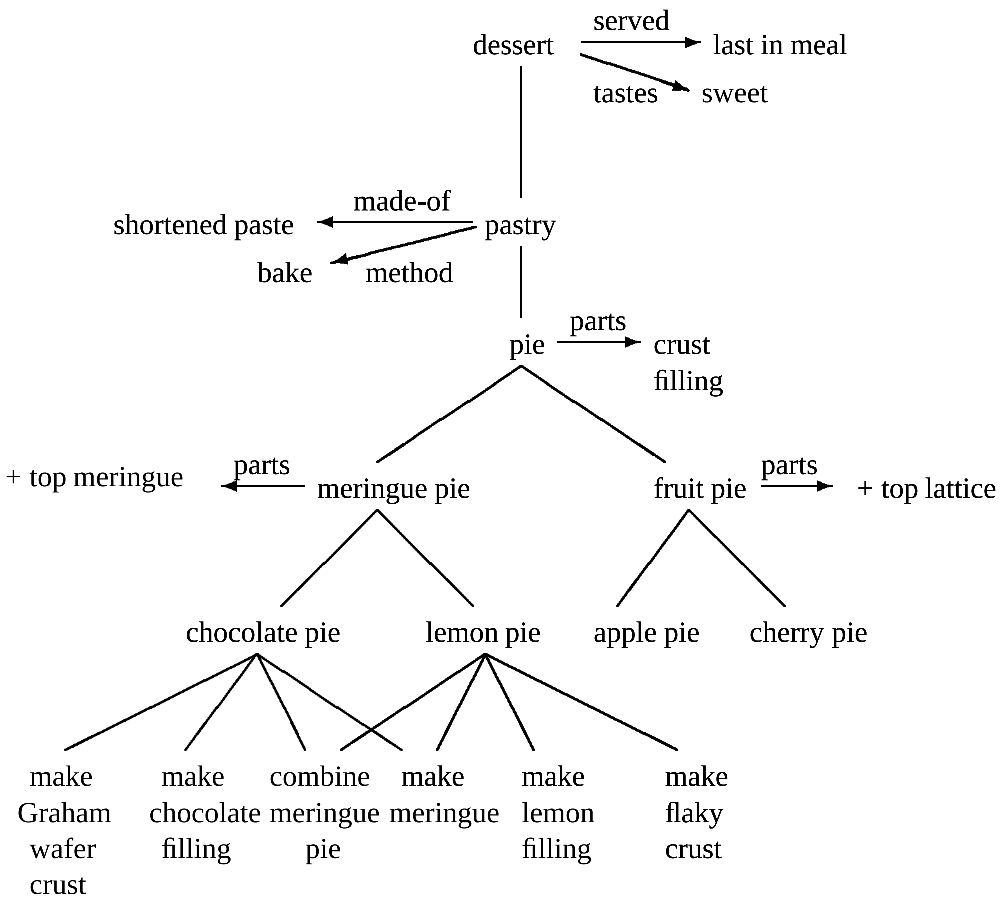


Figure 5.4: Relevant Knowledge Pool for the Question *What is Meringue Pie?*

relevant knowledge pool will look like figure 5.5.

Up to now, we have not made use of the user's domain knowledge to help make decisions in the generation process. The second step, assigning importance values, requires the user's knowledge recorded in the user model to be taken into account. We take the relevant knowledge pool created in step one, and consult the knowledge base in the user model to find out which part of the knowledge pool is already known to the user. For

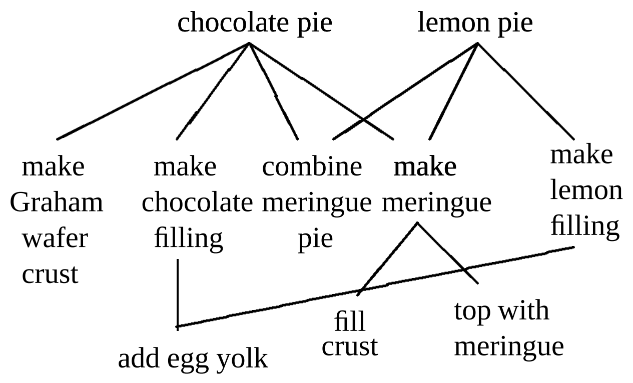


Figure 5.5: Relevant Knowledge Pool for the Question *How to Make Chocolate Pie?*

every node and every arc we assign a value of either 1 or 0^6, indicating that it is known or unknown to the user, respectively. This procedure helps further processes decide which part of the relevant knowledge pool will be used as a reminder, and which will be new information to be provided to the user.

The relevant knowledge pool shown in figure 5.4 and the user's knowledge base in figure 5.3 result in the valued relevant knowledge pool in figure 5.6, where the 1's denote the nodes and links known to the user, and the 0's unknowns.

[6] It is also possible to use values between 0 and 1, indicating that the user knows part of a particular concept. But this value is very difficult to obtain, for different users might have different standards for these fractions, and it is also hard for the user to say how much he/she knows without obtaining complete information about that concept. Therefore, we choose to use the two extreme values only, making the situation easier to implement.

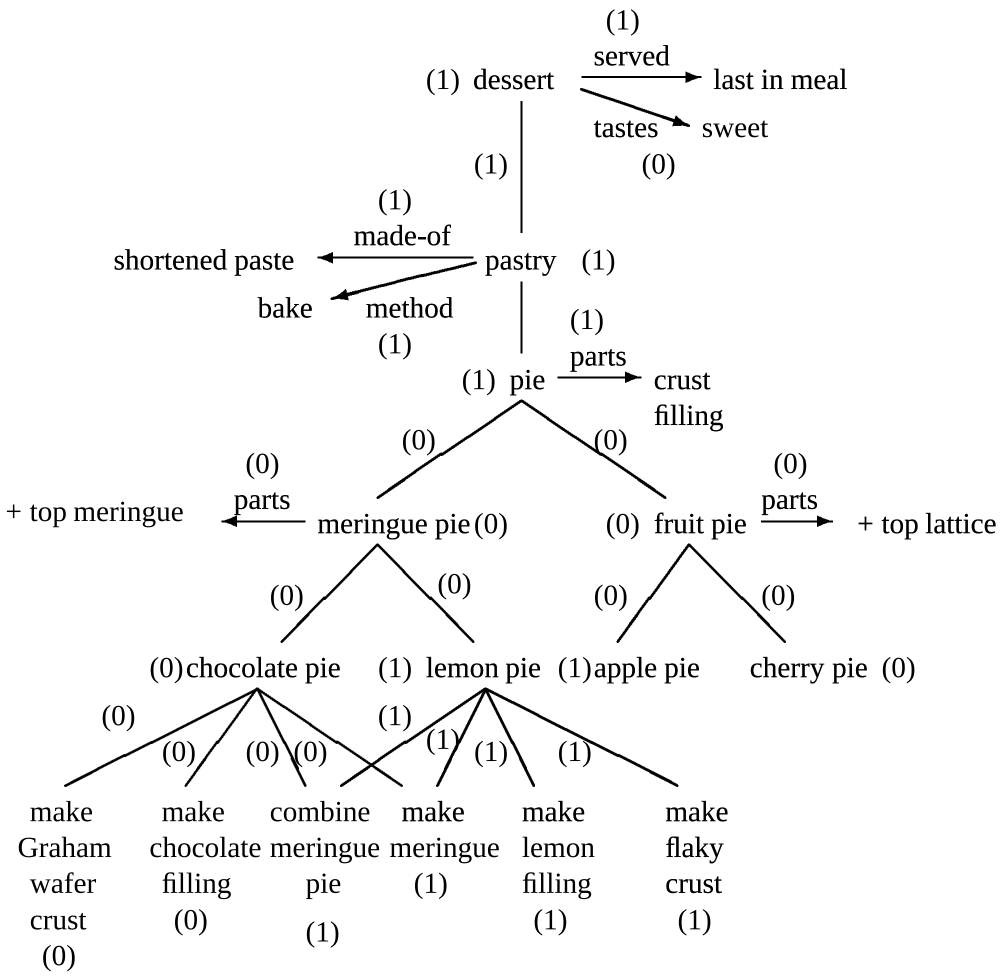


Figure 5.6: Valued Relevant Knowledge Pool

5.4.3 Select Strategy

In section 4.1, the principles of how TEXT selects a particular schema for a given question are discussed. Although the question type and the amount of information available are the main factors for schema selection, there are still other attributes that contribute to making the decision, such as the user's preferences, receptivity and memory capability.

When the amount of information available is at the two extremes for the candidate schemata of a particular question type, the decision can be easily made. Problems arise when they have a similar amount of information. In these situations, we suggest that the decisions are dependent on the individual users. The user's preference for a particular style can help make decisions when the candidates differ in discourse strategies. For example, for definitional questions in TEXT, a person whose preference is attributive will prefer the identification schema, while a person who prefers examples will be given the constituency schema. In addition to preference, a user's receptivity and memory capability are also important. A person with high receptivity and high memory capability is expected to be willing to accept new information and to remember it for a longer period of time. Thus, we can choose to provide more information for this type of users, even by using two schemata.

5.4.4 Select Proposition

In TEXT, the proposition selection is done by instantiating all the candidate predicates (all the outgoing arcs from the current state in the ATN) and applying focus constraints to choose amongst them, as described in section 4.1. The motivation is to make the discourse coherent, which we agree is very important in natural language generation. On the other hand, we also believe that there are more factors that influence proposition selection than simply focus constraints, especially when more than one predicate falls into the category of the highest priority. In this section, we will discuss how information in the user model interacts with the generation process by dividing the proposition selection process into two subprocesses: *predicate instantiation* and *predicate selection*.

5.4.4.1 Predicate Instantiation

This step has to be performed before predicate selection because how the predicate is instantiated determines how the focus will be shifted. Usually, there is more than one way of instantiating a predicate, so this process is not as easy as searching through the database to find information that matches the requirement. It requires more supporting information to decide which one to choose, and this information comes from the user model.

The user's interest, of course, is an important factor that should be considered. This influence takes place when, for example, we are instantiating a particular-illustration predicate. Suppose the text generated so far is *A meringue pie is a kind of pie with meringue on top. Lemon pie and chocolate pie are instances of meringue pies*. Now we have two objects for further illustration, lemon pie and chocolate pie. If the user knows neither of them, a good way of choosing one over the other is to choose the one the user is interested in, if one exists.

The user's knowledge and memory capability can also aid in decision-making. In the above case, if the user knows only lemon pie and has a good memory, we can easily decide that we want to introduce chocolate pie to him/her, since most likely he/she remembers what a lemon pie is, while if the user has a bad memory, we might want to re-introduce lemon pie.

5.4.4.2 Predicate Selection

After instantiating all the possible predicates, what we have to do is to choose one that makes the discourse coherent and suits the user's needs well. The former is achieved by considering McKeown's focus constraints priority list; the latter, then, is obtained by taking into consideration various attributes in the user model. The factors in the user model that contribute to the decision-making process at this stage of the generation process include:

- Preferences: if a predicate is in the list of styles that the user prefers, we choose it over the others.
- Memory capability: if the user has a bad memory, we prefer giving reminders to introducing new material.
- Interests: if the focus of the instantiated predicate is something the user is interested in, it has a higher probability of being chosen.
- Receptivity: if a person has high receptivity, we would like to give him/her more information; therefore, a predicate with an arc which leads back to the current state in the ATN is preferred.
- Knowledge: the user's knowledge helps distinguish a reminder from new information, hence, supporting the decision made by the user's memory capability.

It is obvious that many of the rules above contradict each other. Here we only give the readers an idea of how these attributes influence the generation process, and will introduce the schemes we adopt to solve the problem later in chapter 6.

5.5 Updating the User Model

The main purpose for including user modeling in natural language generation is to tailor the output text to individual users. In order to do so, it is very important that the information in the user model is up to date. In Moore's PEA system ([Moore, 1989]), the user model is not updated as the dialogue progresses, but, instead, the utterances recorded in the dialogue history are used as a source of additional information that the user may know. In this case, the system will be able to know what the user has learned in the current conversation, but not what he/she had in previous interactions. This, of course, is not satisfactory. What we would like to do is to have the system maintain the latest information about the user, thus providing the most suitable responses.

5.5.1 What Should Be Updated?

By analyzing the characteristics of each of the attributes recorded in the user model, we find that some of them are more *static* and others more *dynamic*. The most dynamic one is the user's knowledge, which changes every time the system provides new information to the user. This attribute, of course, has to be updated very frequently, otherwise it is likely that the system will repeat what was just said but has not yet been recorded in the model.

The user's preferences will also change, but more gradually. As pointed out by Paris ([1988]), the same object is described very differently in an adult encyclopaedia than in a junior encyclopaedia, therefore, suggesting that things should be expressed in different styles to expert users and to naive users (here we are not suggesting that adults are experts and juniors are novices, but only that adults are generally more experienced than juniors). If a user is a novice to begin with, and moves towards the expert's extreme as he/she learns from the system, the system can explain things to this user based on what he/she learned, instead of describing everything from scratch as for a naive user. This causes a change in the style of presentation, but we know that this information can be updated much less frequently.

The other attributes recorded in the user model, the user's role, interests, memory capability, and receptivity, can change as well, but even less frequently than the user's preferences. We have no idea when these changes will take place; for example, we do not know when a person will change jobs or when a user gets tired of his/her favourite dish. Therefore, it is difficult for the system to know when the information should be updated.

5.5.2 How to Update Them?

First of all, we will deal with the most important issue, updating the user's knowledge. We believe that the information the user has learned must be recorded after each utterance, since we do not want the system to repeat itself within the same conversation unless the user asks it to do so. But, on the other hand, it is unnecessary and too costly to update the user model every time the system provides the user with some information. An alternative is to record the changes in the relevant knowledge pool and update the user's knowledge in the user model all at once when the topic is changed or when the conversation is over[7]. Since everything mentioned is chosen from the relevant knowledge pool, we guarantee that what should be updated in the user model also exists in the relevant knowledge pool.

With the valued relevant knowledge pool shown in figure 5.6, if the question *what is*

[7]Here we make the assumption that the user understands all the information the system provides. In reality, this might not be true. In ThUMS ([Dent *et al.*, 1987]), the facts in the system are classified into two classes, *simple* facts, those that can be understood and used by the user if conveyed to him/her, or *complex* facts, those that cannot be completely explained by the system. Updating the user model could make use of this distinction to decide which facts should be recorded in the user model and which ones should not. This improves the system's ability to model the user's domain knowledge, and is left for future work.

a meringue pie is asked, the system might answer:

> Meringue pie is a kind of pie. It has meringue on top. There are two instances of meringue pie, lemon pie and chocolate pie. Chocolate pie has the same meringue as lemon pie, but has Graham wafer crust and chocolate filling. Fruit pie is also a kind of pie, but it has top-lattice on top. Apple pie is an instance of fruit pie. Meringue pie and fruit pie both have crusts and fillings, but have different tops.

After this information is provided to the user, the system updates the valued relevant knowledge pool by marking all the nodes and arcs mentioned in the response as known. In this example, the updated valued relevant knowledge pool is as shown in figure 5.7.

Now if the user asks to end the conversation, the system will record all the changes made (the nodes and arcs marked in the relevant knowledge pool) in the user model. All the newly introduced objects will be added to the user model as classes, tokens, subplans or primitive actions. An existing entity can also be updated if new information is provided to the user, for instance, adding an attribute or a child link. This updating process not only adds information to the user model, but also corrects the different hierarchical structure in the user's knowledge base due to his/her *incomplete information*. In this example, the user originally considered *lemon pie* and *apple pie* as children of *pie* (figure 5.3). After asking the above question, the user found out that lemon pie is indeed a child of *meringue pie* and apple pie a child of *fruit pie*. Now the system has to add the two classes *meringue pie* and *fruit pie*, which were unknown to the user, and reorganize the hierarchy by updating the parent-child links in these classes as well.

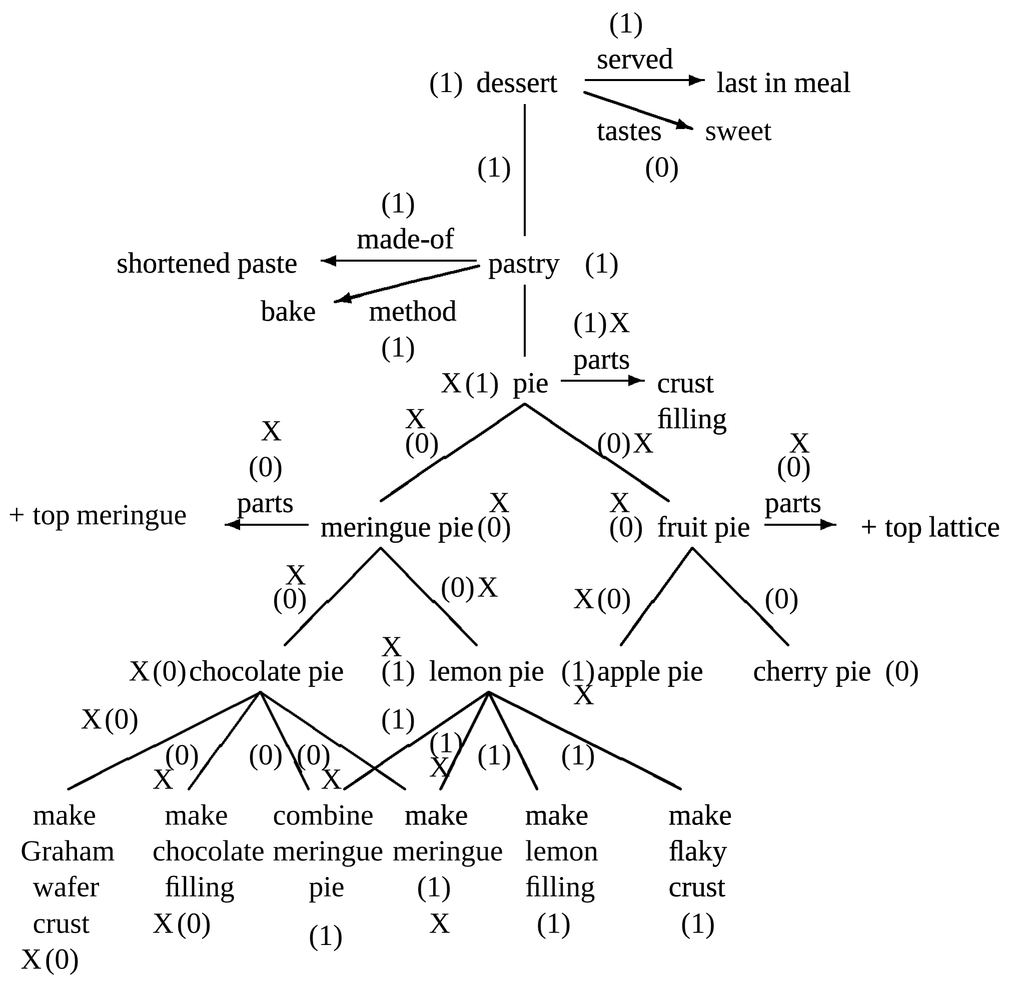


Figure 5.7: Updated Valued Relevant Knowledge Pool

Chapter 6

The Algorithm

In this chapter, we introduce an algorithm for generating responses used in our proposed model. The algorithm will be presented at an abstract level in section 6.1, and in the following sections, further details and discussions will be provided. Also, a more detailed algorithm for each step will be presented at the end of each section.

6.1 An Overview of the Algorithm

Input: question asked by the user.

Output: response tailored to the user according to information available in the system knowledge base and the user model.

Steps:

Repeat steps 1 - 4 until end of conversation.

1. Determine question type (see section 6.2).

```
2. Determine relevant knowledge pool (see section 6.3).

   2.1 Determine global focus.

   2.2 Section off knowledge relevant to the global focus from the database.

   2.3 Consult user model to assign importance value to each node and link.

3. Select Schema(ta) (see section 6.4).

   3.1 Check user's receptivity and memory,
             if both high,
                X = 2 (X denotes the number of schemata to be chosen);
             otherwise,
                X = 1.

   3.2 Determine candidate schemata according to the input question type.

   3.3 Choose X schema(ta) that are the most suitable, if possible (details given in
       section 6.4).

4. Repeat this step until a pop arc is chosen.

   4.1 Instantiate predicates (see section 6.5).

      4.1.1 Determine candidate predicates : all outgoing arcs from current state in
            the ATN.

      4.1.2 Instantiate all the candidate predicates using the rules provided for each
            predicate (rules described in section 6.5).

   4.2 Select predicates (see section 6.6).

      4.2.1 Rank all the instantiated predicates according to,
                 focus constraints
                 user's knowledge
```

user's preference

user's interest

user's receptivity

user's memory capability

4.2.2 Choose the first predicate in the ranked list.

4.3 Update relevant knowledge pool (see section 6.7).

5. Update the user model (see section 6.7).

6.2 Determine Question Type

In section 5.4.1, we explained how the user's role influences what he/she expects the answer to be, therefore helping the system decide the type of the question being asked. A *how* question is obviously a *procedural* question, but how a *what* question is classified is closely related to the global focus and the user's role. Since a procedural answer can only be provided for objects at certain levels in the hierarchy (those that actually have procedures associated with them, which, in our cooking domain, are entity types *token*, *subplan* and *primitive action*), if the global focus happens to be at one of those levels and the user is a person who is more interested in knowing the procedure of the object, it will be classified as a *procedural* question; otherwise, it will be considered as a *definitional* question.

Algorithm:

1. **How** questions: procedural questions.

2. **What** questions:

```
If user prefers procedures,
   if global focus has procedures attached,
      procedural questions;
   otherwise,
      definitional question.
else /∗ user prefers descriptions or user's preference unknown ∗/
   definitional question.
```

The above algorithm, when applied to our cooking domain, will be implemented as:

1. **How** questions: procedural questions.

2. **What** questions:

```
If user's role is a chef,
   if global focus is a token, subplan, or primitive action,
      procedural questions;
   otherwise,
      definitional question.
else /∗ user is a diner or user's role unknown ∗/
   definitional questions.
```

6.3 Determine Relevant Knowledge Pool

The topic has already been discussed in section 5.4.2. The important issues here are to determine the global focus, which is the object being asked about, and create a relevant knowledge pool to provide information for further processing.

The global focus can be easily located in the system's knowledge base using the set of indices. But as discussed earlier, the scope of the relevant knowledge pool depends on the question type. For a definitional question, it will be the global focus's own attributes, its descendants to the subplan level, its ancestors to the root, its siblings and the siblings' children. Choosing the descendants is for the use of the constituency schema, which gives examples of the subclasses or tokens that are included in the object being asked about. The ancestors and their attributes are included mainly for the identification schema which identifies the object as a more specific type of another class (one of its ancestors), and gives the attributes of the object to distinguish it from other children of the same ancestor. The relevant knowledge pool also consists of the global focus's siblings and their children because the compare and contrast schema might be used. When searching for an object to be compared with the global focus, we would like to choose something that is similar but different, which most likely will appear as its sibling in the hierarchy.

For a procedural question, the relevant knowledge pool will be the subtree with the global focus as root, and the parents of all the subplans and primitives actions included in that subtree. We choose the whole subtree so that any level of detail of the recipe can be provided. The parents of the subplans and primitive actions are also included because for a user with a good memory, we can leave out the details of a subplan that he/she knows by saying that the subplan is the same as the one used in another particular dish.

Once the relevant knowledge is sectioned off, the *root* of the relevant knowledge pool is searched in the user's knowledge base via the indices. If it is not found, the root is marked as unknown and its children play the role of the roots recursively until all the root nodes are found or the relevant knowledge pool is entirely searched. If a node in the user's knowledge base is found, the remaining part of the relevant knowledge pool and the corresponding part in the user's knowledge base will be traversed and compared. This process is to find out which parts of the relevant knowledge pool are already known to the user and which are not. This information is also recorded in the relevant knowledge pool and will be used to make decisions in later steps in the generation process.

Algorithm:

1. Determine global focus: global focus = the object being asked about in the question.
2. Determine relevant knowledge: if the question is a definitional question, do step 2.1; otherwise, go to step 2.2.

   2.1 The relevant knowledge pool for a definitional question. For each node found, the record (which represents that object, including its attributes, links, etc.) is added to the relevant knowledge pool.

   2.1.1 Search for the global focus in the system's database through the indices.

   2.1.2 For the ancestors, follow the *parent* links recursively until the second level from the top[1] in the hierarchy is reached (for the parent link in each entity type, see discussion in section 5.2).

   2.1.3 For the descendants, follow all the *child* links recursively until the sub-plan level is reached.

   2.1.4 For the siblings, search for the global focus's parent through the *parent* link and locate the siblings through the parent's *child* links.

   2.1.5 For the siblings' children, search via the siblings' *child* links.

   2.2 The relevant knowledge pool for a procedural question. Again, for each node visited, the record is added to the relevant knowledge pool.

   2.2.1 Search for the global focus in the system's database through the indices.

[1]Generally we include all the ancestors up to the root. Choosing the second level from the root is an additional constraint for our domain as explained in chapter 5.

2.2.2 For the subtree, follow the *child* links and the children's *child* links recursively until the primitive actions are reached.

2.2.3 For all the subplans and primitive actions included in the subtree, follow their *parent* links for all the nodes (tokens and subplans for subplans and primitive actions, respectively) that share the same action.

3. Assign importance values:

   3.1 Locate the root of the relevant knowledge pool in the user's knowledge base.

   3.2 If not found, assign 0 to the root and its children links, and use its children as roots. Repeat steps 3.1 and 3.2 until all the root nodes are found or the relevant knowledge pool is searched through.

   3.3 Traverse and compare the relevant knowledge pool from the root(s) and the user's knowledge base from the corresponding node(s) found in previous steps.

   3.4 For each node and link in the relevant knowledge pool, assign 1 to it if it also appears in the user's knowledge base. Assign 0 otherwise.

6.4 Select Schema

We inherit McKeown's idea of using schemata as discourse patterns to help generate text for different purposes, but apply them differently so as to generate text in a broader scope, both in the *amount* and the *style* of the information provided. In this section, we introduce the schemata used in our system, and discuss the criterion we use to choose amongst them.

6.4.1 Schemata Available

The number of schemata available determines the number of styles the text can have. Obviously, the more schemata we have, the more varieties of text can be generated.

In McKeown's TEXT system, there are four schemata available, *identification*, *constituency*, *attributive*, and *contrastive*, as mentioned in chapter 2. Though any question can be answered in one of these four different forms, the answers all provide *static* information, i.e., there are no *temporal* relations in the information provided. This severely restricts the possible application domains for the system. In order to overcome this problem, we add one schema that handles *procedural* questions, which, in our cooking domain, are questions regarding how to make a particular dish.

We adapt three schemata used in TEXT for answering definitional questions, and modify Paris's process-trace schema ([Paris, 1987]) for our procedural schema. Generally speaking, the schemata are domain independent, but since the type of information varies from domain to domain, some predicates that are applicable in one domain may not be suitable in others. Therefore, we will make some modifications to serve the particular needs of our domain.

Figure 6.1 shows the ATN graph for our identification schema. It first identifies the object, describes it, then gives some examples to further explain it. The last two stages in McKeown's original schema, shown in figure 2.3, contain mostly predicates that appeared in earlier stages. Since we will show some examples of tracing our algorithm later, we leave out the last two stages in our ATN to present reasonably small illustrations. We also omit the analogy predicate in the description subroutine since a contrastive schema can be used together with the identification schema, which provides a similar effect with more details.

The ATN graph for the modified constituency schema is shown in figure 6.2. This

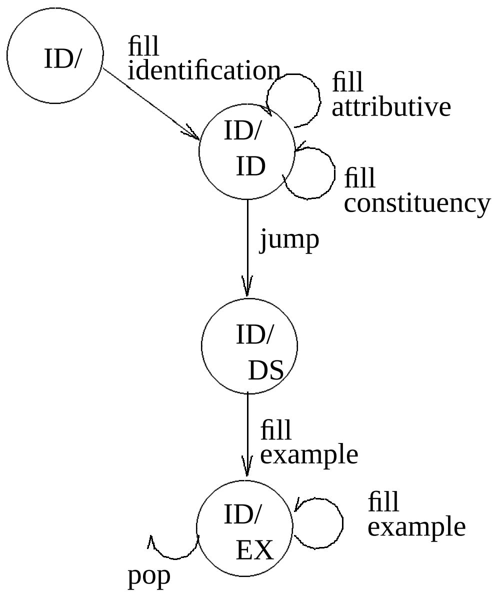


Figure 6.1: ATN for the identification schema

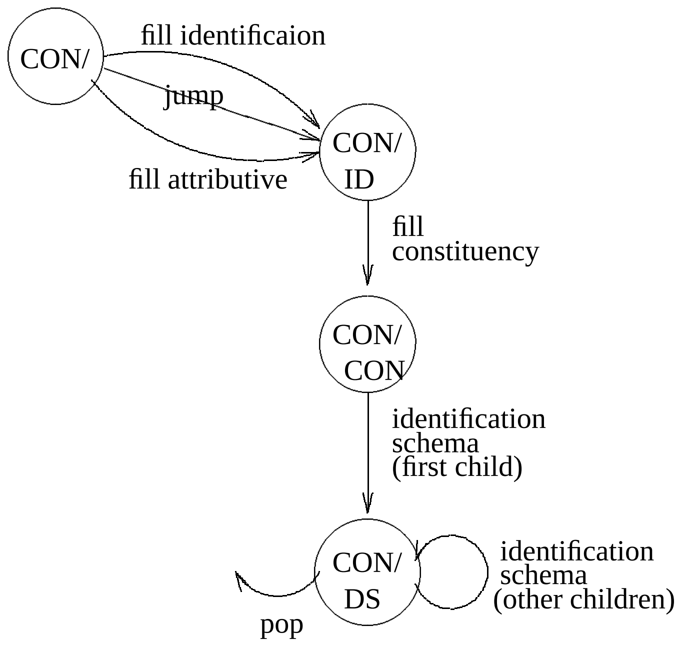


Figure 6.2: ATN for the constituency schema

version of the constituency schema is less complicated than the one used in TEXT. This schema is based on the same schema used by Paris ([1988]), which gives a brief description of the object being asked about, lists its children, and describes them in turn.

We show our contrastive schema in figure 6.3, which differs considerably from the same schema in TEXT because of the different usage of the schema. In TEXT, the contrastive schema is used when the user asks about the differences between two objects, while in our system, it is chosen as a supporting schema for a definitional question[2]. S-ince in our system, this schema is always instantiated after a portion of text has been

[2]The *contrastive* schema will not be used alone because question types are limited in our study. But for questions like *what is the difference between X and Y*, the contrastive schema will be needed. For more about how this schema is used alone, see [McKeown, 1985].

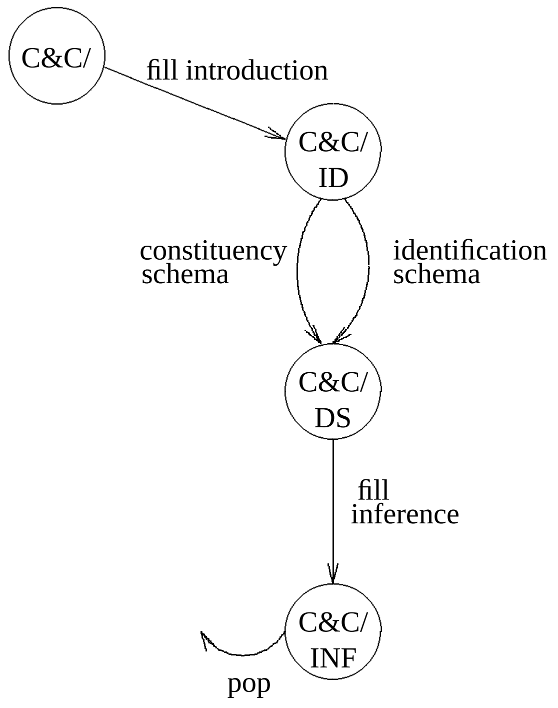


Figure 6.3: ATN for the contrastive schema

generated by either the identification schema or the constituency schema, one of the objects has already been introduced in detail. Therefore, what is left to be done for the contrastive schema is to introduce the object to be compared with (using the identification or constituency schema), and to include some of the similarities and differences between the two.

The procedural schema is designed to give a description of how an object works or how to perform a particular task. In our cooking domain, it gives a step-by-step instruction of how to make a particular dish. The ATN graph for the schema is presented in figure 6.4. It gives a brief introduction to the global focus, followed by the steps for performing the task, with further illustrations of detailed sub-steps, if necessary.

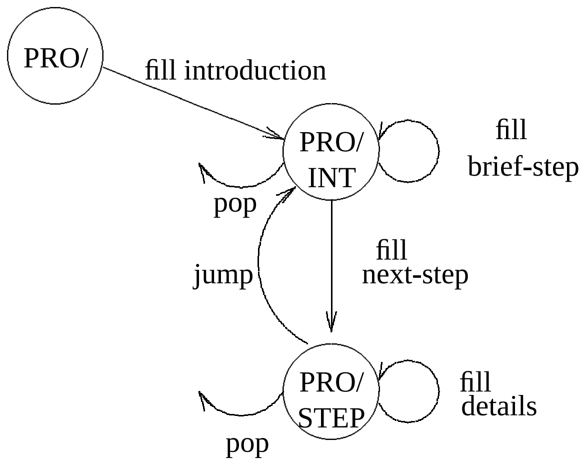


Figure 6.4: ATN for the procedural schema

6.4.2 The Procedure

As in TEXT, we also have some pre-determined candidate schemata associated with both question types, listed in figure 6.5. For the definitional questions, in addition to the two original schemata used in TEXT, we also add the contrastive schema. It is designed to be used for people with high receptivity and a good memory, to provide them with more information by comparing the object asked with another similar one. A constraint in using this schema is that it can only be used as a second schema in the response, i.e., either the identification or constituency schema has to be chosen and instantiated before it.

As shown in the algorithm, we suggest that for a person with both high receptivity and a good memory, we will provide a richer response by using two schemata if possible. We choose this constraint because a person with high receptivity is willing to learn more, and a person with a good memory will be able to remember more. We predict that only under these circumstances will the additional information provided be worthwhile[3].

[3]We decide this based on our intuition, instead of studying actual occurring texts for the patterns. For

Definitional questions

  Identification

  Constituency

  Contrastive

Procedural questions

  Procedural

Figure 6.5: The Pre-determined Schemata for Both Question Types

Now that we have all the candidate schemata, what we have to do is to choose 'X' schema(ta) among them (X determined as in the algorithm). We will discuss this issue by dividing the situation into the following cases:

Procedural Questions:

This is the simplest case since the only choice is the procedural schema. Here, only one schema will be used no matter what the value of X is.

Definitional Questions:

For a definitional question, there are two possibilities:

1. Choosing one schema: It will be either an identification or constituency schema. The criteria for choosing one over the other is basically the same as the one used in TEXT, comparing the amount of information available for each. This is done by checking if the object in focus in the question appears above or below a pre-determined level in the hierarchy[4]. In the former case, the constituency schema will

future work, it would be worthwhile to do some rigorous testing to defend our arguments in a fashion similar to the design methodology of Paris [1987].

[4]In our knowledge base, we choose two levels to represent this, the *token* level and its parent level.

be chosen, while in the latter, the identification schema is preferred. If the object is at the boundary level, we will consult the user's preference in the user model, where *attributes* gives a preference to the identification schema and *examples* to the constituency schema.

2. Choosing two schemata: As discussed before, the first of the two schemata must be either identification or constituency, and the second contrastive. We choose the first as we did in case 1, and decide if the second schema should be included by searching through the relevant knowledge pool for an object to be used in comparison. The object has to be a sibling of the global focus which is known to the user or a sibling with at least one of its children known to the user (this can now be easily done by checking all the siblings of the focus and their children because every node is valued either 1 or 0, according to whether the user knows it or not). If such an object exists, we will adopt the contrastive schema as well.

Algorithm:

1. Determine number of schemata chosen.
   Check user's receptivity and memory,
   ```
           if both high,
              X = 2;
           otherwise,
              X = 1.
   ```

2. Determine candidate schemata according to the question type decided in section 6.2 and the question-schemata mapping in figure 6.5.

This is because in our cooking domain, the nodes appearing at both these levels have a similar amount of information about themselves and their children.

```
3. Choose appropriate number of schemata.

   3.1 If the question type is procedural, choose procedural schema no matter what
       the value of X is.

   3.2 If the question type is definitional,

       3.2.1 First decide whether identification or constituency schema will be used.
             If the global focus appears above the pre-determined level,
               choose the constituency schema;
             if it appears below the pre-determined level,
               choose the identification schema;
             if it appears at the pre-determined level,
               if the user prefers attributes,
                 choose the identification schema;
               otherwise,
                 choose the constituency schema;

       3.2.2 Determine if the contrastive schema will be used.
             If X = 2 and there exists in the relevant knowledge pool a sibling of the
             focus, either itself or one of its children is known to the user,
               choose the compare and contrast schema.
```

6.5 Instantiate Predicate

In most cases, there is more than one outgoing arc from a state in the ATN, which indicates that there is more than one choice for the next proposition. In order to choose amongst them, the predicates (outgoing arcs) first have to be instantiated.

There are nine different predicates in the four schemata used in our system, each of them providing different styles of information and having different instantiation rules. We will introduce the instantiation rules and give an example for each of them in the following. The examples shown here will be the same as the one used in chapter 5 for the question *what is a meringue pie* (see figures 5.1, 5.3, and 5.6 for the system's database, user's knowledge base, and the valued relevant knowledge pool, respectively).

1. The identification predicate: This predicate identifies the object as a type of its ancestor. Normally we would like to introduce it as a child of its parent, but only if the parent is known to the user. If not, we will continue searching upwards in the hierarchy until we meet an ancestor that the user knows. When the question *what is a meringue pie* is asked, the system identifies the focus (meringue pie) and locates its parent through the *ISA* link in the record for the class *meringue pie*. In this case, the object *pie* is found, and the identification predicate is instantiated as *meringue pie is a kind of pie*, as pie is known to the user[5].

2. The attributive predicate: This predicate introduces the object's attributes to the user, which, in our domain, is suitable for the use of entity types *class* and *token*. In our database, the attributes are associated only with the entity type *class*, and are listed in the *WITH* slot. Note that since property inheritance is one of the important aspects of our domain, a class includes not only attributes recorded specifically within this class, but also those inherited from its ancestors which this class and its sibling classes all share. In some cases, it is important to search for the attributes associated with the ancestors to obtain the complete information. For instance, the

[5]If none of the ancestors of the focus are known to the user, its parent will be chosen and defined. For example, for the same *what is a meringue pie* question, if none of *pie*, *pastry*, and *dessert* is known, the identification predicate will still be instantiated as *meringue pie is a kind of pie*, followed by *pie consists of two parts, the crust and filling*.

only attribute listed under meringue pie is *parts: + top meringue*. The ‘+’ sign here indicates that there are other parts recorded under its ancestors. Searching for the attributes in its parent, we find that the other parts are *crust* and *filling*. The ones that will actually be mentioned depends on the user’s memory capability and his/her knowledge. Those that are already known to the user will not be repeated if the user has a good memory. In the case where the user has a poor memory, the attributive predicate will be instantiated as *meringue pie has three parts, the crust, the filling, and the meringue*.

3. The constituency predicate: The children of the focus will be introduced when the constituency predicate is instantiated. In our domain, it is applicable to all the *classes* and *tokens* when a definitional question is asked. For a *class*, we look under the *CHILDREN* slot for its children, and for a *token*, the *PROCEDURE* slot. The number of children that will be included depends on the entity type of the focus, the user’s receptivity, memory capability, interests, and knowledge. These factors will be discussed in the order they should be considered.

   3.1 Entity type: For an object in the *class* category, its children are the instances of particular types of that class; therefore, any number of them can be mentioned. If the object is a *token*, the children are the subplans for making the dish; therefore, all of them have to be included, otherwise the procedure will not be complete. Hence, if the focus is a token, introduce all children; otherwise, proceed to the next step.

   3.2 User’s receptivity and memory capability: These two factors combined determine the amount of information provided to the user. Obviously, the type of users to whom we would like to provide the most information are those having both high receptivity and a good memory; therefore, for these users, we will introduce three children[6]. The users that will be provided the least information are those that have both low receptivity and low memory capability, and will be introduced only one child. In the other cases, two children will be mentioned.

3.3 User's knowledge and memory capability: These are used to decide whether old or new information should be provided. We will assign *weights*, which are all initialized to zero, to the children of the focus. These weights indicate the priority for choosing children (the higher ones preferred). We will increase by one the weight values of the children that are totally unknown to the user if the user has a good memory and of those that are completely known if he/she has a poor memory. For the children that are partially known to the user (by *partially known* we mean that the child *node* is known, but the *link* between the object and the child is unknown), we increase their weights by 0.5. The above process indicates that new information is preferred if the user has a good memory; otherwise, a reminder is preferred. (This will be referred to as the *memory and knowledge principle* in the algorithm presented later in this section)

3.4 User's interests: This also contributes to deciding which children to choose by adjusting the weight for each of them. This rule suggests that we increase by one the weight values of each child that appears in the user's interest list. (This will be referred to as the *interest principle* in the algorithm)

Having decided upon the number of children to be included in the predicate and having evaluated all the children, we can easily choose the number of children we

[6]We believe that three children is a reasonable amount to mention in one proposition because we do not want to enumerate all the children at one time. The issue of the number of children to be mentioned under different circumstances is still an open question, and is left for future study.

want from the beginning of the ranked list. In the case where a tie occurs, we give preference to the user's memory capability (combined with the user's knowledge), since we believe that it is better to provide the user with information that he/she is not particularly interested in, than to tell him what he/she is interested in but already knows. If they happen to have the same preference according to the user's memory capability, we will arbitrarily pick one of them.

4. The example predicate: This predicate illustrates in detail a particular child of the current focus. First of all, we have to decide which child to choose. A weight-assigning strategy is used again, this time considering the user's knowledge, interests, and memory capability.

   4.1 If the user has a good memory, add one to the weight values of the children unknown to the user.

   4.2 If the user has a poor memory, add one to the ones known to the user.

   4.3 Add one to the children in the user's interest list.

   Once we have the ranking of the preferences of each child, the one ranked highest will be chosen as the focus of the example predicate. If a tie occurs, we prefer the memory capability over interests because of the same reason given in point 3 above. Again, if this does not solve the problem, an arbitrary one will be chosen.

   Since the purpose of the example predicate is to introduce an object to the user, it can be thought of as giving a simplified answer to a new definitional question with the new object as focus. Thus, we can choose to introduce its attributes (*attributive* predicate), its children (*constituency* predicate), or to compare it with its siblings (*inference* predicate). The preferences are given below.

   4.1 In the case where the user has a good memory, if one of the new focus's siblings is known to the user, and they share some children, then we would like to compare the two of them since this provides the user with new knowledge by integrating the new information with his/her old knowledge, giving him/her a better picture of the knowledge base. In this case, the sibling will be mentioned and the parts that they share will be introduced, as well as the other characteristics specific to the new focus.

4.2 If the new focus is not qualified in case 4.1, we will choose either the attributive predicate or constituency predicate according to the the user's preference. If there is no information available for the predicate that the user prefers, then choose the other predicate.

In the *what is a meringue pie* example shown in chapter 5, the third sentence of the sample output on page 70 is the constituency predicate, introducing the two children of meringue pie. The following sentence, instantiating the example predicate to be *inference,* further illustrates chocolate pie by comparing it with lemon pie which is in the user's knowledge base.

5. The inference predicate: This is used in the contrastive schema, concluding the comparison of the two objects. This includes the similarities and dissimilarities between them. The common attributes shared by the two objects can be found in their parent node, and the different ones are listed under each of them separately. For example, in the meringue pie example, after introducing fruit pie as another instance of pie, the inference *meringue pie and fruit pie both have crusts and fillings, but have different tops* is made.

6. The introduction predicate: The introduction predicate is used as the first step in the procedural schema to provide a brief description of the object being asked about. This is done by giving the user the outline of how the task can be performed, i.e.

by providing all the subplans under a token or the primitive actions under a subplan. For example, when the question *how is a meringue pie made* is asked, the introduction predicate will be instantiated as *to make meringue pie, you have to make the crust, filling, and meringue separately, then combine them*.

7. The brief-step predicate: This predicate is used to briefly introduce the next subplan in a token. If the subplan is known to the user, the name of the subplan, as well as an arbitrary token in the relevant knowledge pool that shares the same subplan, will be introduced; otherwise, only the name of the subplan will be mentioned.

8. The next-step predicate: This predicate is used for describing the next step in the process in more detail. If the focus is a *token* with subplans, the *next-step* predicate, evoked repeatedly, describes more specifically the subplan introduced by the last *brief-step* predicate. If it is a *token* without subplans, a *subplan*, or a *primitive action*, it gives a description of the next step to be performed, which is recorded in its *PROCEDURE* slot.

9. The details predicate: This predicate is called when the step introduced by the *next-step* predicate needs to be further explained. This will be the case when a primitive action is included in a subplan. When instantiating the *next-step* predicate, if the information includes a lower-level action that can be further illustrated, the *details* predicate will be instantiated, unless the lower-level action is known to the user and the user has a good memory.

   For example, when the question *how is flaky pie crust made* is asked, the first *next-step* predicate is instantiated as *sift flour and salt into a bowl, cut shortening into flour*. Since the primitive action *cut shortening* is included in this step, it can be further expanded as *cut shortening into flour with a pastry blender or two knives until no pieces of shortening larger than a pea appear, when the bowl is shaken.*

Algorithm:

Check for the type of the next predicate.

Case 1: identification:

1. Search for the focus's ancestors in a bottom-up way until one that the user knows is found.
2. If an ancestor is found in step 1, introduce the focus as a type of that ancestor.
3. Otherwise, introduce the focus as a type of its parent and define the parent.

Case 2: attributive:

1. Search for the focus's attributes in its own entry.
2. Search for relevant attributes in the ancestor nodes, if necessary[7].
3. Choose the attributes using the *memory and knowledge principle*[8].
4. Introduce all the attributes that have non-zero weights.

[7]Here the *relevant* attributes are defined as the attributes with the same attribute type. All the attribute values with the same type are prefixed with a '+' sign except the one occurring at the highest level in the hierarchy. For example, the attribute *parts* for meringue pie has value + *top meringue*; therefore, its parent will be searched where the remaining values *crust* and *filling* are found.

[8]The *memory and knowledge principle* was described in point 3.3 earlier in this section. To summarize, it means,

Weight = 1 if either the object is unknown to the user and the user has a good memory,
or the object is known to the user and the user has a poor memory;
= 0.5 if the object is partially known to the user;
= 0 otherwise.

Case 3: constituency:

1. Search for all the children of the current focus.
2. If the focus is of entity type *token* or *subplan*, introduce three children (X = 3).
3. If it is of type *class*,

   3.1 If the user has receptivity and memory both high,
       introduce all children;
     if the user has one of them high,
       introduce two children (X = 2);
     otherwise,
       introduce one child (X = 1).

   3.2 If not all children chosen, assign weights to all children using the *memory and knowledge principle* and the *interest principle*[9].

   3.3 Choose the top X child(ren) according to the evaluation scheme in step 3.2.

Case 4: example:

1. Determine which child to further illustrate.

   1.1 Evaluate each child using the *memory and knowledge principle* and the *interest principle*.

   1.2 Choose the child ranked the highest, and let it be the new focus.

[9]The *interest principle* was described in point 3.4 earlier in this section. To summarize, it means,

Weight = 1 if the object is in the user's interest list;
= 0 otherwise.

2. Determine how to illustrate it.

```
If the new focus's sibling is known and shares common child(ren) with the focus,
and the user has a good memory,
   choose the *inference* predicate;
otherwise,
   if the user prefers attributes,
      choose the *attributive* predicate.
   if the user prefers examples,
      choose the *constituency* predicate.
   if there is no information available for the predicate the user prefers,
      choose the other predicate.
```

Case 5: inference:

1. Search for the similarities between two objects in their parent node.
2. Search for the dissimilarities within individual nodes.
3. Introduce both similarities and dissimilarities.

Case 6: introduction:

1. Search for the focus's children.
2. Introduce all the children as parts of performing the procedure.

Case 7: brief-step:

If the object is a *token*, and has *subplans* associated with its procedures in the *PROCEDURE* slot,

```
 if the next subplan is known to the user and the user has a good memory,
   mention the subplan and an arbitrary token known to the user that shares the same
   subplan,
 otherwise,
   mention the subplan briefly by name;
```

Case 8: next-step:

```
If the object is a token with subplans,
  if there still exist steps to be performed in the last introduced subplan,
    introduce the next step in the subplan's PROCEDURE slot;
if the object is a token without subplans,
  introduce the next step to be performed in the token's PROCEDURE slot;
if the object is a subplan or a primitive action,
  introduce the next step to be performed in the PROCEDURE slot of the
  subplan or the DESCRIPTION slot of the primitive action.
```

Case 9: details:

1. Search for the primitive action mentioned in the last *next-step* predicate.
2. Illustrate that primitive action by providing the details in the *DESCRIPTION* slot.

6.6 Select Predicate

After the candidate predicates are instantiated, we have to decide which one of them will be chosen as the next proposition. The predicate selection process, as discussed in section 5.4.4.2, is influenced by a number of factors recorded in the user model, such as the user's preferences, memory capability, interests, receptivity, and knowledge. In addition to those factors, the focus constraints adopted by TEXT will also be taken into consideration.

Obviously, some of the factors listed above contradict each other. For example, the user's interests suggest that we talk about something that the user is interested in, and the user's memory capability suggests that we mention information new to the user if he/she has a good memory. In this case, if the user has a good memory and already knows about what he/she is interested in, conflicts arise between the interest and memory capability factors. Also, a user who prefers attributive information may have interests falling into the category of information given by examples (the focus's children). Here, the preference factor suggests that attributive information be chosen, while the interest factor prefers examples. In order to solve these problems, we adopt a *weighting strategy* once again, this time using different weights for different factors, according to the importance of each of them. Thus, the ultimate decision is made by considering the combined, rather than the individual, effect of the six factors, in order to solve the problems of contradiction mentioned above.

In the following passage, we discuss the factors that contribute to the predicate selection process for a definitional question, both the way they influence the decision and their importance, i.e. the amount they contribute to the total weight values. Here, we use W(fc), W(pre), W(int), W(mc), and W(rec) to represent the weight values assigned to a particular predicate by the factors focus constraints, preferences, interests, memory capability, and receptivity, respectively.

1. Focus constraints: the focus constraints influence the coherence of the output text by trying to restrict the focus shifts between sentences to a certain range. According to the priorities in McKeown's focus constraint list (figure 2.4), we should prefer

either shifting the focus to something just introduced or maintaining the same focus, where the former has a higher priority. Therefore, we let W(fc) = 1.5 if the focus of the instantiated predicate would shift to an object just introduced, W(fc) = 1 if the focus remains the same, otherwise, W(fc) = 0.

2. User's preferences: this factor gives the user's preference of the style of the predicate, being either *attributes* or *examples*. The preference *attributes* corresponds to the *attributive* predicate, while *examples* includes both *constituency* and *example* predicates. For all the predicates that fall into the user's preference list, we let W(pre) = 1, and for the others, W(pre) = 0.

3. User's interests: though this factor influences predicate instantiation more than predicate selection, it still has some contribution to the selection process. It happens especially when two predicates have equal weights for the other factors. In this case, the predicate in the user's interest list will be chosen, if one exists. Since this factor makes a minor contribution, we will assign W(int) = 0.5 if the focus is in the interest list, and W(int) = 0 if it is not.

4. User's receptivity: this influences the amount of information provided to the user. It is straightforward that by choosing the arcs in the ATN that lead to the states which are already traversed longer output will result, therefore providing more information to the user. Hence, we let W(rec) = 1 if the user has high receptivity and the arc of a predicate in the ATN leads to a traversed state, or if the user has low receptivity and the arc leads to a new state. Otherwise, we let W(rec) = 0.

5. User's knowledge: the user's knowledge itself does not influence the decision process directly. It only determines which part of the relevant knowledge pool is already known to the user and will be used as a reminder. Therefore, it is used in combination with the user's memory capability attribute.

6. User's memory capability: in order to avoid the possible conflicts that might arise between the user's receptivity and memory capability, we further restrict the memory capability factor to influence the amount of *reminders* provided. For a person with a good memory, we do not want to repeat old information; therefore, we let W(mc) = 1 if the information provided is completely unknown, W(mc) = 0.5 if it is partially known, and W(mc) = 0 if it is completely known, indicating that new information is preferred. On the other hand, for a user with a poor memory, we let W(mc) = 1 if the information is known, W(mc) = 0.5 if it is partially known, and W(mc) = 0 if it is unknown, suggesting that reminders are chosen over new information.

In the cases where ties occur, we give the preferences as:

memory constraints $>$ receptivity $>$ focus shifts $>$ preferences $>$ interests

We consider *providing the user with appropriate information* the most important task; therefore, the user's knowledge and memory constraints (combined as W(mc)) should be at the top of the list. The user's receptivity concerns the *amount* of information to be provided which should be ranked the second, followed by the focus shifts, guaranteeing the coherence of the output text. Between the last two attributes, preferences dominates interests simply because the latter contributes less to the total weight than the former (W(int) = 0.5 if focus is in the interest list).

Here we assign the weight value for a *pop* arc to be 1.5. Therefore, if we are at a final state where all the outgoing arcs have values less than 1.5, the pop arc will be chosen, which means ending the output. This is intended to prevent long and meaningless output for users with high receptivity. Since the largest value that can be assigned to a predicate is 4.5 (all having 1's, except interest being 0.5), and the most common values that appear are 2 and 2.5, we consider a predicate with value 1.5 as providing *non-critical* information. Therefore 1.5 is chosen as the threshold value when a pop arc is encountered.

Take the same user's knowledge base shown in figure 5.3 as an example. Suppose the other attributes in the user model are,

preference :	attributive
interest :	chocolate
receptivity :	high
memory :	high
role :	diner

When the question *what is a meringue pie* is asked, the *identification* and *contrastive* schemata are chosen according to the previous steps in the algorithm. For the identification schema instantiation, the first predicate, *identification*, is instantiated as *meringue pie is a kind of pie*. The second predicate, however, has three possible choices, according to the ATN shown in figure 6.1. The three choices, the *attributive*, *constituency*, and *example* predicates[10], are instantiated and evaluated as,

1. Attributive

   It has meringue on top

W(fc) = 1	focus = meringue pie
W(pre) = 1	user prefers attributes
W(int) = 0	
W(rec) = 1	user receptivity high, and leads back to the same state
W(mc) = 1	user memory high, and attribute unknown

   W(Att) = sum of all above = 4

[10]According to the ATN, the third choice should be the *jump* arc. Since it is not possible to evaluate a jump arc, we evaluate all the outgoing arcs from the state that the jump arc leads to, which in this case is the *example* predicate.

2. Constituency

   There are two instances of meringue pie, lemon pie and chocolate pie.

W(fc) = 1	focus = meringue pie
W(pre) = 0	
W(int) = 0.5	user is interested in chocolate
W(rec) = 1	user receptivity high, and leads back to the same state
W(mc) = 0.5	nodes known (lemon and chocolate pies), but arcs unknown (the relationship between meringue pies and lemon and chocolate pies)

   W(Con) = sum of all above = 3

3. Example

   Chocolate pie has the same meringue as lemon pie, but has Graham wafer crust and chocolate filling

W(fc) = 0	focus = chocolate pie
W(pre) = 0	
W(int) = 0.5	user is interested in chocolate
W(rec) = 0	user receptivity high, but leads to a new state
W(mc) = 1	user memory high, and information unknown

   W(Ex) = sum of all above = 1.5

From the assigned weight we find that the attributive predicate is the most preferred; therefore, it is chosen as the second proposition.

For a procedural question, there exist fewer variations because the basic procedure has to be entirely introduced. We choose to vary the level of detail of the description according to the user's domain knowledge. According to the ATN graph for the procedural schema shown in figure 6.4, when in the state *PRO/INT*, the three choices are *brief-step*, *next-step*, and *pop*. *Brief-step* is chosen when there exists a subplan not yet introduced (e.g. is always chosen to initially describe a subplan), *next-step* is selected when the current subplan is not completely described, and *pop* is followed if the whole procedure is explained. In the state *PRO/STEP*, there are also three candidate arcs, *fill details*, *jump* and *pop*. We follow *fill details* when the last *next-step* predicate introduces a primitive action not known to the user, we choose the *jump* arc if the procedure is not yet completed (in order to choose *next-step* again), and we select the *pop* arc, again, if the whole procedure has been explained.

Algorithm: For a definitional question,
For all the candidate predicates,

1. Assign to each of them a value W(fc), where

   W(fc) = 1.5, if the candidate's focus is something newly introduced;
   1, if it remains the same focus;
   0, otherwise.

2. Assign to each of them a value W(pre), where

   W(pre) = 1, if the predicate appears in the user's preference list;
   0, otherwise.

3. Assign to each of them a value W(int), where

   W(int) = 0.5, if the focus appears in the user's interest list;
   0, otherwise.

4. Assign to each of them a value W(rec), where

W(rec) = 1, if the user has high receptivity and the arc leads back
to the same state,
or if the user has low receptivity and the arc proceeds
in the graph;
0, otherwise.

5. Assign to each of them a value W(mc) according to the *memory and knowledge principle* (see footnote 2 on page 93).

6. Add up all the values for each predicate and choose the one with the highest value as the next proposition.

For a procedural question,

1. If in *start* state, follow the only outgoing arc.

2. If in *PRO/INT*,
   choose *next-step* if the current subplan is only specified by name and is not yet completely introduced;
   choose *brief-step* if there is another subplan to be described;
   choose *pop* otherwise.

3. If in *PRO/STEP*,
   choose *details* if the last *next-step* predicate introduces a primitive action not known to the user;
   choose *jump* if the current subplan is not yet completely introduced;
   choose *pop* otherwise.

6.7 Update the User's Knowledge

In section 5.5 we discussed *what* should be updated in the user model and *how* it should be updated. We also agreed that since some of the attributes in the user model are more dynamic and some more static, they differ in how frequently they should be updated.

The attribute that should be updated most frequently is the user's knowledge, which changes every time the system provides information to the user. This attribute is updated after every conversation, when either the user asks to end the conversation or the topic is changed to something beyond the scope of the current relevant knowledge pool. During the conversation, all the new information provided to the user is recorded in the relevant knowledge pool by marking the nodes and links mentioned. After the conversation, all the changes recorded in the relevant knowledge pool will be used to update the knowledge base in the user model. The changes in the user model can be *adding a new node* (introducing a new object), *filling in slots of an existing node* (providing new attributes or children), or even *updating existing slots* (correcting a user's incomplete information). We will give an example to show how the process works.

First, we start off with the user's knowledge base, for example, the one shown in figure 5.3. Here the user does not know anything about *meringue pie* and *fruit pie*; therefore, *lemon pie* and *apple pie* are considered as children of *pie*. This is a case of *incomplete information*. In order to correct the different hierarchy in the user's knowledge base, links in two places should be updated. The original links *pie – lemon pie* and *pie – apple pie* will be broken, and two sets of new links *pie – meringue pie*, *meringue pie – lemon pie* and *pie – fruit pie*, *fruit pie – apple pie* will be added. For the question *what is a meringue pie*, the relevant knowledge pool is shown in figure 5.4. By consulting the user's knowledge base, we get the valued relevant knowledge pool as presented in figure 5.6.

When the first proposition in the output *meringue pie is a kind of pie* is generated, the

```
CLASS pie
    ISA pastry
    WITH
        parts : crust, filling
    CHILDREN
        lemon pie
        apple pie
END
```

Figure 6.6: User's Original Knowledge Base

node *meringue pie* and the link between *pie* and *meringue pie* in the relevant knowledge pool will be marked. After the second proposition *it has meringue on top*, the attribute *parts : + top meringue* will be marked as known. Thus, continuing until the last proposition, we get the updated valued relevant knowledge pool as shown in figure 5.7.

Now, if the user asks to end the conversation, these updated values in the relevant knowledge pool should be recorded in the user model. We first show part of the original knowledge base of the user (in its actual representation as discussed in section 5.2) in figure 6.6, and the corresponding part of the updated knowledge base in figure 6.7. We can see that in the partial knowledge base shown, one new node is added (*meringue pie*), slots are filled in (*CHILDREN : lemon pie*, *chocolate pie* for *meringue pie*), and some slots are updated (*CHILDREN : meringue pie, fruit pie* for *pie*).

The other attributes in the user model are much more *static* than the user's knowledge, and there is usually no way for the system to know when these values will change. Therefore, the update of these attributes are *passive*, i.e. the system will not make any changes to them unless the user asked it to do so.

```
CLASS pie                    CLASS meringue pie
    ISA pastry                   ISA pie
    WITH                         WITH
        parts : crust, filling       parts : + top meringue
    CHILDREN                     CHILDREN
        meringue pie                 lemon pie
        fruit pie                    chocolate pie
END                          END
```

Figure 6.7: User's Updated Knowledge Base

Algorithm for updating the relevant knowledge pool:

1. Search via the indices in the relevant knowledge pool for all the nodes and links mentioned in the last proposition.
2. Mark them as known.

Algorithm for updating knowledge in the user model:

1. For all the nodes and links marked in the relevant knowledge pool, search for the corresponding entries in the knowledge base in the user model using the indices.
2. Add a new node if a new object is introduced.
3. Fill in slots of an existing node if new attributes or new children are introduced.
4. Update existing slots if the user had incomplete information before.

Chapter 7

Examples

In this chapter, we will show some predicted outputs of our proposed model. Since the model is not implemented, the outputs are obtained by hand-tracing through the algorithm presented in the previous chapter. In this chapter, we will only show the choices made at major decision points because the trace processes are long and tedious. However, we will present the details for example 1 in appendix B to further clarify the actual process of the algorithm for interested readers.

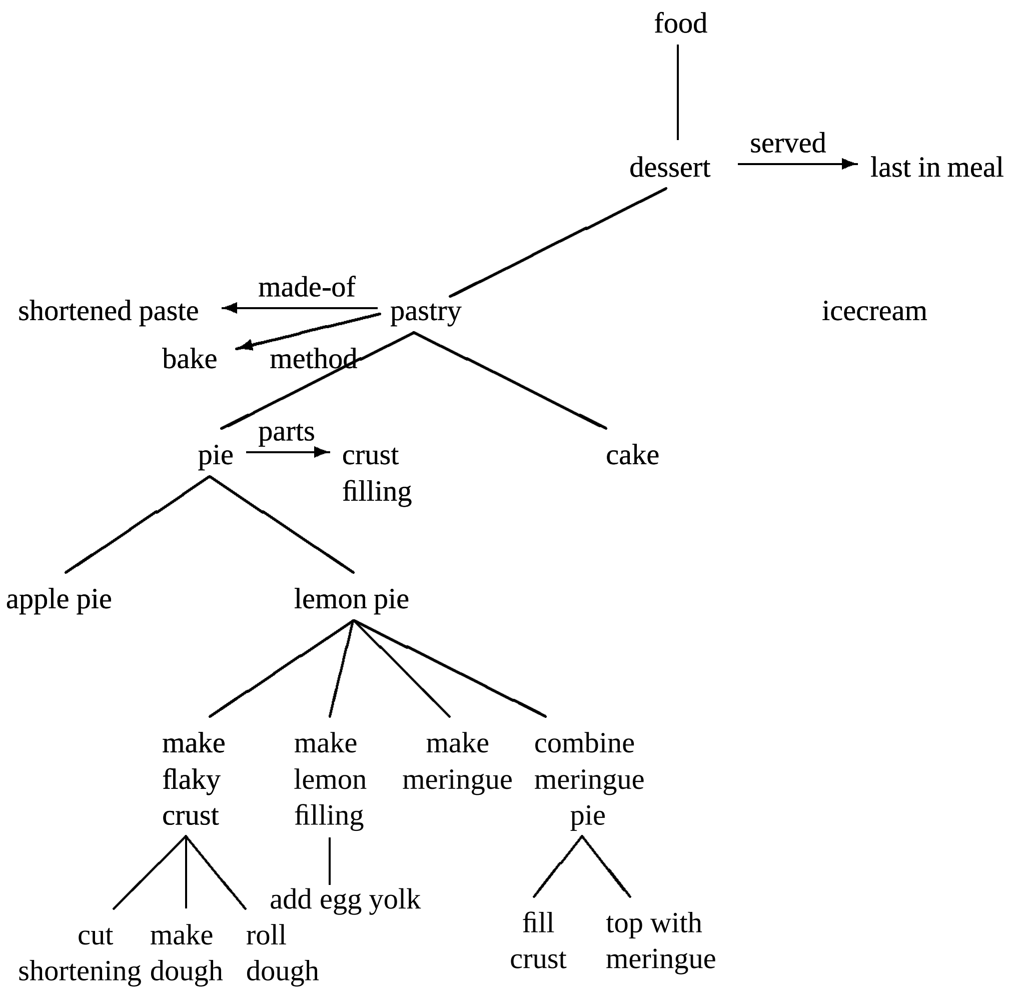
Example 1

In this example we show how the sample output for the question *what is a meringue pie* in section 5.5 is derived.

We consider the partial cooking database in figure 5.1 and a user with knowledge shown in figure 5.3. The user's knowledge with other attributes in the user model are shown again in figure 7.1 for convenience.

When the question *what is a meringue pie* is asked by this user, the algorithm will be traced as follows.

1. Knowledge:



2. Interest: chocolate

3. Preference: attributes

4. Receptivity: high

5. Memory capability: high

6. Role: diner

Figure 7.1: User Model in Example 1

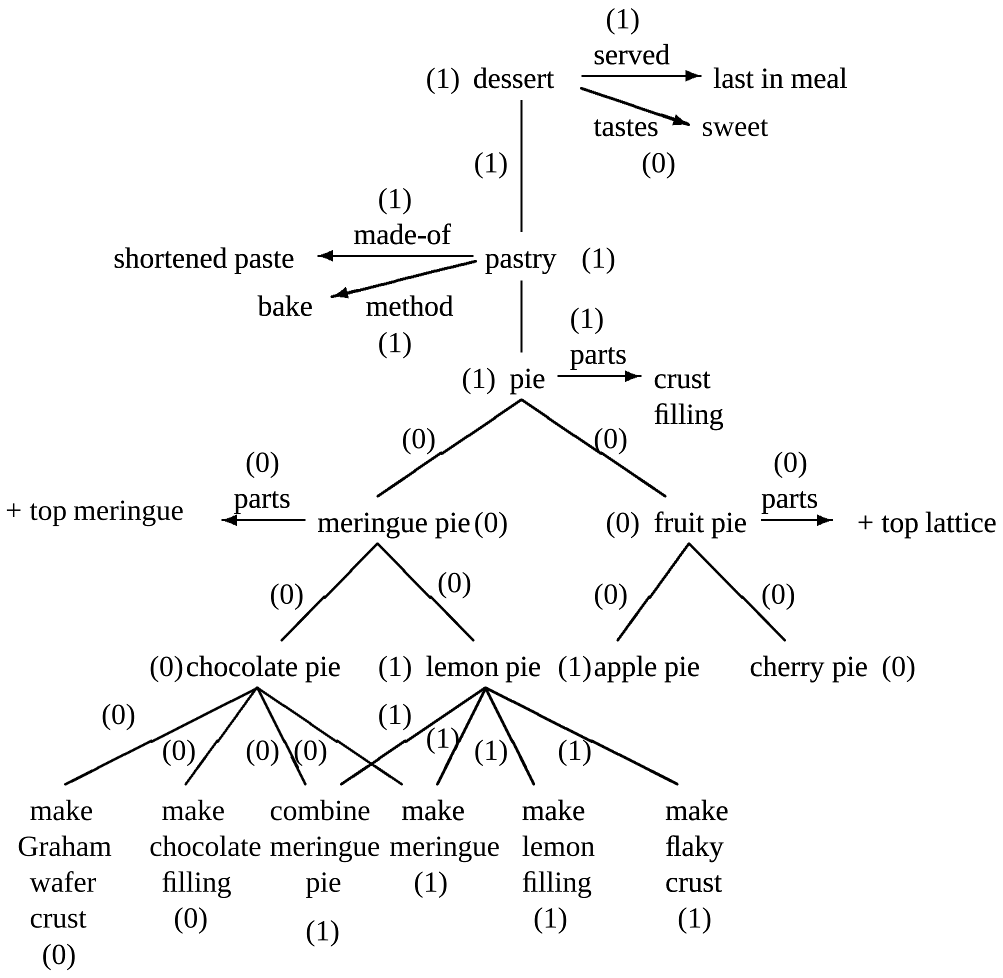


Figure 7.2: Relevant Knowledge Pool in Example 1

Step 1: Determine question type

This is a *what* question, and since the user is a diner, it is categorized as a *definitional* question.

Step 2: Determine relevant knowledge pool

In this question, the global focus is *meringue pie*. According to the rules presented in section 6.3 for choosing relevant information and by consulting the user's knowledge in the user model, we get the valued relevant knowledge pool shown in figure 7.2.

Step 3: Select schemata

Since the user has both high receptivity and a good memory, the number of schemata chosen (X) is two. In this case, the two schemata chosen are the *identification* schema and the *contrastive* schema (details provided in appendix B).

Step 4: Select propositions (including instantiate predicates and select predicates)

The identification schema is first instantiated, followed by the contrastive schema. The object to be compared in the contrastive schema is *fruit pie* since it is a sibling of meringue pie and one of its children, *apple pie*, is known to the user. The states visited and the arcs chosen in the ATNs are shown in the following (for the ATNs, see figures 6.1 and 6.3):

Identification schema:

ID/	– identification	$\longrightarrow$ ID/ID
	– attributive	$\longrightarrow$ ID/ID
	– constituency	$\longrightarrow$ ID/ID
	– jump	$\longrightarrow$ ID/DS
	– example	$\longrightarrow$ ID/EX
	– pop	$\longrightarrow$ END

Contrastive schema:

C&C/	– identification	$\longrightarrow$ C&C/ID
	– attributive	$\longrightarrow$ C&C/DS
	– constituency	$\longrightarrow$ C&C/DS
	– inference	$\longrightarrow$ C&C/INF
	– pop	$\longrightarrow$ END

The final output for this question, according to the predicates chosen above, is shown in figure 7.3.

> 1. Meringue pie is a kind of pie. 2. It has meringue on top. 3. There are two instances of meringue pie, lemon pie and chocolate pie. 4. Chocolate pie has the same meringue as lemon pie but has Graham wafer crust and chocolate filling. 5. Fruit pie is also a kind of pie. 6. It has top-lattice on top. 7. Apple pie is an instance of fruit pie. 8. Meringue pie and fruit pie both have crusts and fillings, but have different tops.

Figure 7.3: Output for Example 1

Example 2

Here we have a user with user model shown in figure 7.4. Note that this user has exactly the same information in the user model as the user in example 1, except that he/she has a poor memory.

Since the user's role and knowledge are both the same as in example 1, the first two steps *determine question type* and *determine relevant knowledge pool* will be the same as presented before.

Step 3: Select schemata

The number of schemata chosen is one, because of the user's low memory capability. The two candidates are the *identification* schema and the *constituency* schema, where the former is chosen because the user prefers attributes.

Step 4: Select propositions

The ATN of the identification schema will be traversed in the following order, which generates the output shown in figure 7.5.

$$\begin{array}{lll} \text{ID/} & \text{– identification} & \longrightarrow \text{ID/ID} \\ & \text{– attributive} & \longrightarrow \text{ID/ID} \end{array}$$

1. Knowledge: same as example 1
2. Interest: chocolate
3. Preference: attributes
4. Receptivity: high
5. Memory capability: low
6. Role: diner

Figure 7.4: User Model in Example 2

1. Meringue pie is a kind of pie. 2. In addition to the crust and filling, it has meringue on top. 3. There are two instances of meringue pie, lemon pie and chocolate pie. 4. Lemon pie consists of flaky pie crust, lemon filling and meringue.

Figure 7.5: Output for Example 2

– constituency	$\longrightarrow$ ID/ID
– jump	$\longrightarrow$ ID/DS
– example	$\longrightarrow$ ID/EX
– pop	$\longrightarrow$ END

In this example, we see that much less information is provided to the user than in example 1, due to the user's poor memory. Also we find that, though the traversal of the ATN for the identification schema is exactly the same is in example 1, the outputs are different because of the influence of the memory attribute at the predicate instantiation

1. Knowledge: same as example 1
2. Interest: chocolate
3. Preference: attributes
4. Receptivity: low
5. Memory capability: high
6. Role: diner

Figure 7.6: User Model in Example 3

stage. The second proposition provides more information, where the crust and filling are used as a reminder. Moreover, the last sentence focuses on something the user already knows, therefore ensuring that the system has a correct view of the user's knowledge base before any further information might be provided.

Example 3

In this example, the user model is presented in figure 7.6, where the only difference with the one in example 1 is the user's low receptivity.

As in example 2, the question type is still classified as a *definitional question*, and the relevant knowledge pool is also the same. Furthermore, the schema chosen is the *identification* schema as well, this time due to the user's low receptivity.

Step 4: Select propositions

In this case, the ATN will be traversed differently, as presented in the following. The output for this user is shown in figure 7.7.

1. Meringue pie is a kind of pie. 2. It has meringue on top. 3. Chocolate pie has the same meringue as lemon pie, but has Graham wafer crust and chocolate filling.

Figure 7.7: Output for Example 3

ID/	– identification	$\longrightarrow$	ID/ID
	– attributive	$\longrightarrow$	ID/ID
	– jump	$\longrightarrow$	ID/DS
	– example	$\longrightarrow$	ID/EX
	– pop	$\longrightarrow$	END

In this case, even less information is provided than in example 2. This is because of the fact that the user has low receptivity, preventing us from providing too much new information, and that the user has a good memory, making reminders unnecessary. The ATN traversal is different from that in example 2 by choosing the *example* predicate immediately after the *attributive* predicate. When evaluating the two predicates *constituency* and *example* in example 2, the weights assigned to each of them are shown as follows:

Constituency : *There are two instances of meringue pie, lemon pie and chocolate pie*.

W(fc)	= 1;	focus = meringue pie
W(pre)	= 0;	
W(int)	= 0.5;	user is interested in chocolate
W(rec)	= 1;	user receptive high, and back to the same state
W(mc)	= 0;	user memory low, and information unknown

W(Con) = sum of all above = 2.5

Example	: *Lemon pie consists of flaky pie crust, lemon filling and meringue*.	
W(fc)	= 0;	focus = lemon pie
W(pre)	= 0;	
W(int)	= 0;	
W(rec)	= 0;	user receptive high, but leads to a new state
W(mc)	= 1;	user memory low, and information known
W(Ex)	= sum of all above = 1	

Therefore, the constituency predicate is chosen. While in example 3, the same two predicates are evaluated differently as follows:

Constituency	: *There are two instances of meringue pie, lemon pie and chocolate pie*.	
W(fc)	= 1;	focus = meringue pie
W(pre)	= 0;	
W(int)	= 0.5;	user is interested in chocolate
W(rec)	= 0;	user receptive low, but back to the same state
W(mc)	= 1;	user memory high, and information unknown
W(Con)	= sum of all above = 2.5	

Example	: *Chocolate pie has the same meringue as lemon pie, but has Graham wafer crust and chocolate filling*.	
W(fc)	= 0;	focus = chocolate pie
W(pre)	= 0;	
W(int)	= 0.5;	user is interested in chocolate

1. Knowledge: same as example 1
2. Interest: chocolate
3. Preference: attributes
4. Receptivity: low
5. Memory capability: low
6. Role: diner

Figure 7.8: User Model in Example 4

W(rec)	= 1;	user receptive low, and leads to a new state
W(mc)	= 1;	user memory high, and information unknown
W(Ex)	= sum of all above = 2.5	

Here a tie occurs. Since the two predicates also have the same weights assigned for the first preference user's knowledge (used together with memory constraints), according to the second choice, user's receptivity, we choose the example predicate over constituency.

Example 4

Now we change both the user's receptivity and memory capability to low and examine the result. The user model is given in figure 7.8.

Still, the schema chosen is the *identification* schema, and the relevant knowledge pool is the same as shown in figure 7.2. The ATN is traversed in the following order, with the output presented in figure 7.9.

1. Meringue pie is a kind of pie. 2. In addition to the crust and filling, it has meringue on top. 3. Lemon pie consists of flaky pie crust, lemon filling and meringue.

Figure 7.9: Output for Example 4

ID/	– identification	$\longrightarrow$ ID/ID
	– attributive	$\longrightarrow$ ID/ID
	– jump	$\longrightarrow$ ID/DS
	– example	$\longrightarrow$ ID/EX
	– pop	$\longrightarrow$ END

Though the traversal of the ATN is the same as that in the previous example, the generated texts are very different. Little new information is provided because of the user's low receptivity, and both sentences two and three are instantiated to be mainly used as reminders.

In the above examples, we showed how the user's *receptivity* and *memory capability* influence the behaviour of the generation process. In the next example, we demonstrate how the user's *preference* influences the output text.

Example 5

Though there are two candidate schemata for the definitional question type in examples 1 - 4, the one used is always the *identification* schema, because of the user's preference in *attributes*. In this example, we present a user who prefers *examples* and show how the output is different from the previous ones. The user model for this example is shown in figure 7.10.

The first two steps are still the same, since the user has the same role and knowledge.

1. Knowledge: same as example 1
2. Interest: chocolate
3. Preference: examples
4. Receptivity: high
5. Memory capability: low
6. Role: diner

Figure 7.10: User Model in Example 5

The *what* question is still classified as a *definitional* question with the relevant knowledge pool shown in figure 7.2.

Step 3: Select schema

In this case, the two candidate schemata are still the identification schema and the constituency schema. Since the user's preference is *examples*, the latter will be chosen.

Step 4: Select propositions

The ATN for the constituency schema (figure 6.2) will be traversed in the following order, which generates the output text as shown in figure 7.11.

$$\begin{array}{rll}
\text{CON/}- & \text{attributive} & \longrightarrow \text{CON/ID} \\
- & \text{constituency} & \longrightarrow \text{CON/CON} \\
- & \text{id\_schema} & \longrightarrow \text{CON/DS} \\
- & \text{id\_schema} & \longrightarrow \text{CON/DS} \\
- & \text{pop} & \longrightarrow \text{END}
\end{array}$$

> 1. Meringue pie has crust, filling, and meringue on top. 2. It has two children, lemon pie and chocolate pie. 3. Lemon pie consists of flaky pie crust, lemon filling and meringue. 4. Chocolate pie has the same meringue as lemon pie, but has Graham wafer crust and chocolate filling.

Figure 7.11: Output for Example 5

In this example, we see that more examples are introduced than the previous ones using the identification schema, because of the user's preference in examples. The lemon pie example is given first because it is used as a reminder, which is considered more important than the user's interest, chocolate pie, due to the user's poor memory.

Example 6
In this example, we change the user's role from a *diner* to a *chef* and see how this factor influences the generation process. The user model is shown in figure 7.12, where the user is a chef, and has low receptivity and a good memory.

When this user asks the question *what is a chocolate pie*, the algorithm works considerably differently from the previous examples.

Step 1: Determine question type
Though this is also a *what* question, it is classified as a *procedural* question because the user is a chef.

Step 2: Determine relevant knowledge pool
According to the discussion in section 6.3, the relevant knowledge pool of a procedural question is defined to be the subtree with the global focus, in this case *chocolate pie*, as root and the parents of all the subplans and primitive actions included in the subtree. This relevant information, together with the importance values for each node and arc, are

1. Knowledge: same as example 1
2. Interest: chocolate
3. Preference: examples
4. Receptivity: low
5. Memory capability: high
6. Role: chef

Figure 7.12: User Model in Example 6

shown in figure 7.13.

Step 3: Determine schema

As presented in figure 6.5, the only schema available for a procedural question is the *procedural* schema.

Step 4: Select propositions

For this question, the ATN for the procedural schema is used (figure 6.4). The arcs chosen and the states visited are shown in the following, and the output is shown in figure 7.14.

PRO/	– introduction	$\longrightarrow$ PRO/INT
	– brief-step	$\longrightarrow$ PRO/STEP
	– next-step	$\longrightarrow$ PRO/STEP
	– jump	$\longrightarrow$ PRO/INT
	– next-step	$\longrightarrow$ PRO/STEP
	– jump	$\longrightarrow$ PRO/INT
	– next-step	$\longrightarrow$ PRO/STEP

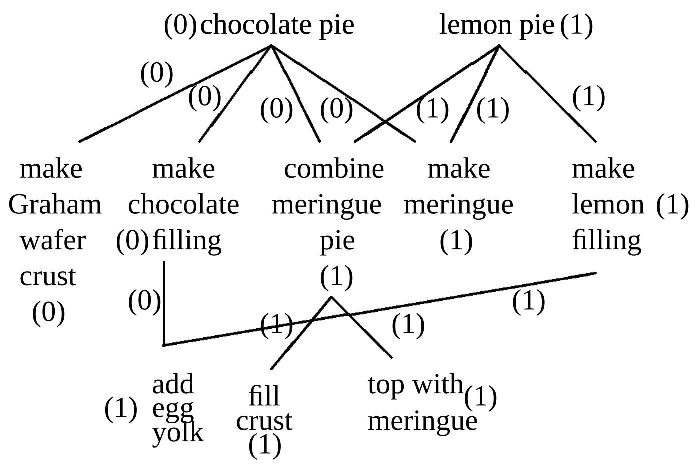


Figure 7.13: Relevant Knowledge Pool in Example 6

– jump	$\longrightarrow$ PRO/INT
– brief-step	$\longrightarrow$ PRO/STEP
– next-step	$\longrightarrow$ PRO/STEP
– jump	$\longrightarrow$ PRO/INT
– next-step	$\longrightarrow$ PRO/STEP
– jump	$\longrightarrow$ PRO/INT
– next-step	$\longrightarrow$ PRO/STEP
– jump	$\longrightarrow$ PRO/INT
– brief-step	$\longrightarrow$ PRO/STEP
– brief-step	$\longrightarrow$ PRO/STEP
– pop	$\longrightarrow$ END

In the output text we see that the subplans for making chocolate pie are provided first, followed by the steps in detail. The first two subplans, *make Graham wafer crust* and *making chocolate filling*, are first introduced using the *brief-step* predicate, then described by using the *next-step* predicate repeatedly. This is because both of the subplans are

1. To make chocolate pie, you have to prepare the crust, filling, and meringue separately, then combine them. 2. To make the crust, mix the crumbs and sugar, add butter. Press mixture firmly into the pie plate to a thickness of 0.5 cm. Bake until lightly browned. 3. For the filling, combine flour, sugar and cocoa, add liquid, stir and cook until thick. Beat egg yolks and add into hot mixture. Remove from heat and add butter. 4. The meringue is prepared as that for lemon pie. 5. Combine the crust, filling, and meringue also as for a lemon pie.

Figure 7.14: Output for Example 6

unknown to the user. The primitive action *add egg yolk*[1] is not explained since it is the one used in making lemon pie which is known to the user (in the case that it is not known to the user, the arc *details*, which provides further explanation for a primitive action, will be chosen). The last two subplans, *make meringue* and *combine meringue pie*, are mentioned using the *brief-step* predicate only, because they are exactly the same as used in a lemon pie and are, again, in the user's knowledge and not just specified by name. In general, as we can see in this example, a recipe is given by first mentioning its subplans, followed by describing the subplans in turn. The subplans that are already known will be mentioned briefly using the arc *brief-step* only, while the ones unknown will be expanded using *next-step* repeatedly. The *primitive actions* within the subplans are treated in a similar way. Thus, dividing the recipes into *subplans* and subplans into *primitive actions* prevents the system from providing too much information (describing a subplan the user already knows), and makes the description useful and terse.

[1]A description of how this action is being done is attached to the node for primitive action *add egg yolk*, which says *Beat egg yolks and into them stir a small quantity of the hot mixture. Blend thoroughly and return to the hot mixture; stir until thick*. Descriptions are generally omitted from figure 5.1 for conciseness.

Chapter 8

Related Work

We have given an introduction to some related work in chapters 1 to 3, and have described our work in detail in chapters 4 to 7. In this chapter, we will further discuss several related natural language generators which are relevant to ours and compare them with our model.

8.1 Paris's TAILOR System

Paris ([1987] and [1988]) investigated the problem of how the user's *level of expertise* can influence the generation process. She analyzed a number of texts and claimed that, in order to provide an answer that is optimally informative, not only should the *amount* of information vary, but also the *kind* of information provided should be different.

8.1.1 System Overview

Paris claimed that a system should provide different *kinds* of information depending on the user's level of expertise. For an expert user, the more commonly used discourse strategy is the *constituency schema* (for details, see [McKeown, 1985]), while for a naive user, a *process trace* is used more often. She implemented a system, TAILOR, which operates on this principle. From the TAILOR system architecture depicted in figure 8.1, we can see that it takes an input question, consults the knowledge base and user model, and generates a description of the content of the answer. This description is passed to the dictionary and then to the surface generator to produce the actual English output.

TAILOR operates in a domain that describes complex devices such as telephones, telescopes, etc. Two sample outputs of describing a *microphone* for expert users and naive users are shown in figures 8.2 and 8.3, respectively. As we can see from the two examples, the response for the expert introduces the subparts of a microphone, while that for the novice describes the process of how a microphone works. The difference is caused by the choice of the constituency schema or the process trace for the expert and novice, respectively. In TAILOR, the answers are provided not only for users at the two extremes, but also for those falling anywhere on the whole spectrum. In order to generate appropriate responses for users having *local expertise*, the two strategies have to be mixed, and decisions have to be made within the generation process. Thus, it enables the system to provide answers for users having an intermediate level of expertise by switching between the constituency schema and the process trace depending on the user's local expertise.

8.1.2 The User Model

In Paris's TAILOR system, user modeling is introduced, but she only considers the user's domain knowledge, i.e. the user's *level of expertise*. She records in the user model the user's *knowledge about specific items in the knowledge base* and *knowledge about various basic underlying concepts*, which help determine whether a constituency schema, a process trace, or a combination of both should be used. Note that the decisions made

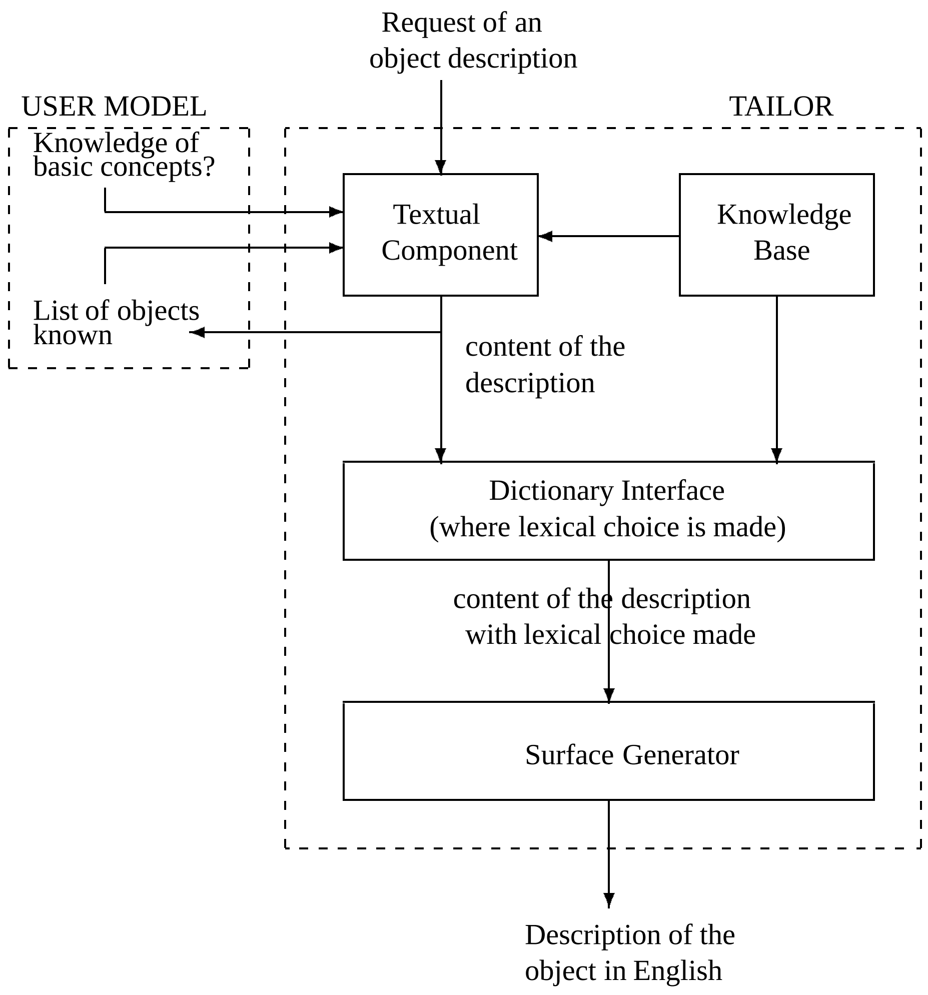

Request of an object description
USER MODEL
TAILOR
Knowledge of basic concepts?
List of objects known
Textual Component
Knowledge Base
content of the description
Dictionary Interface (where lexical choice is made)
content of the description with lexical choice made
Surface Generator
Description of the object in English

Figure 8.1: The TAILOR System Architecture

The microphone is a device that changes soundwaves into current. It has a disc-shaped aluminium diaphragm and a doubly-resonant system to broaden the response. The system has a carbon chamber and an air chamber. The diaphragm is clamped at its edges.

Figure 8.2: Describing a microphone to an expert user

A person speaking into the microphone causes the soundwaves to hit the diaphragm of the microphone. The diaphragm is aluminium and disc-shaped. The soundwaves hitting the diaphragm causes the diaphragm to vibrate. When the intensity of the soundwaves increases, the diaphragm springs forward. This causes the granules of the button to be compressed. The compression of the granules causes the resistance of the granules to decrease. This causes the current to increase. Then, when the intensity decreases, the diaphragm springs backward. This causes the granules to be decompressed. The decompression of the granules causes the resistance to increase. This causes the current to decrease. The vibration of the diaphragm causes the current to vary. The current varies like the intensity varies.

Figure 8.3: Describing a microphone to a naive user

in the generation process depend on the user's domain knowledge, which is explicitly indicated in the user model, as opposed to having default knowledge for each stereotype as in most other existing systems.

TAILOR obtains its information about the user by making inference from various factors. *User type* is considered because some classes of users are more likely to be experts and some novices. Also, classes might be useful in giving *a priori* information on the possible *local expertise*. *Memory organization* plays an important role in deciding that a part of the knowledge base is obscure. If the user is knowledgable about a part deep in the hierarchical tree, TAILOR assumes that he/she has local expertise in that area. *Question type* and *misconceptions* contribute to deciding the user's level of expertise as well. By examining the way the user asks a question, TAILOR infers whether the user knows that concept or not. TAILOR also uses some *inference rules* to determine the user's *radius of expertise*. Particularly, when the user model indicates local expertise about an object, TAILOR assumes that the superordinate and subparts of the object are also known. The last strategy TAILOR uses is to ask the user questions to begin with, and update the user model as the dialogue progresses.

8.1.3 Comparisons

Paris's work is also based on McKeown-style generation, inheriting the idea of using schemata as discourse strategies, and implementing them using ATNs. She added the *process trace* schema which is similar to our *procedural* schema, but for the definitional questions, only the *constituency* schema is used. In our model, we provide more varieties of text for a definitional question by including the *identification*, *constituency*, and *contrastive* schemata.

The major difference between TAILOR and our proposed model is the factors taken into account in the user model. Paris only looked into the problem of how *user's knowledge* influences the content of the description, which is a subset of the factors we consider in our model. We agree with Paris that *user's knowledge* plays an important role in the generation process, but we claim that other aspects also have considerable influence on choosing what to say. Consider the case of a man (not a mechanic) who wants to replace a door lock on his front door. If he asks for a description about door locks, the information he expects to get is *what the subparts of a door lock are*, instead of *how a door lock works*. But since this person is not a mechanic, he might be classified as a naive user. In this case, the *process trace* schema will be chosen, providing him with inappropriate information. In order to solve this problem, we include the *user's role* in the user model as a clue of the user's possible goal, and the *user's preference* indicating the *kind* of information the user prefers. Finally we use the user's knowledge to decide the *amount* of information to be presented. However, we do not consider the influence of user's knowledge on the *style* of presentation, as was done in TAILOR.

In TAILOR's user model, the user's knowledge is represented by pointers pointing to the objects that the user knows about in the knowledge base. It deals with the objects in general, without a detailed indication of which attributes of the object the user knows about. In our model, we have a separate knowledge base indicating the user's domain knowledge in which we include all the nodes and links the user knows about. Therefore, every attribute and subpart of an object will be taken into account, giving a more detailed picture of the user's knowledge.

One important aspect of user modeling in a question answering system is that it should be a long-term process, i.e. it should be able to model a user over a long period of time, instead of a short time interval, such as a single conversation. TAILOR does not address the issue of updating the user model, but focuses on how the user's level of expertise may influence the response. However, it does keep track of the objects just introduced during a conversation and assumes that the user has the local expertise of these objects from then on. The major problem is that this information is only kept during the same conversation, and not recorded permanently. Therefore, when a user consults the system later, the system *forgets* what the user has learned before. This is impractical for a question answering system, since the system doesn't keep an exact record of the user's knowledge. In order to improve this situation, we not only mark off in the relevant knowledge pool all the nodes and links mentioned during the conversation, but also copy this information to the user model after the interaction. Therefore, our user model is able to operate over multiple sessions.

In general, Paris's work is quite similar to ours except that we consider different aspects in the user model and look at the effect of the user's domain knowledge differently. Also Paris combines the two strategies for expert and naive users for a whole range of possible users, which can perhaps be applied to our cooking domain to provide a combination of definitional and procedural information for users having an intermediate level of expertise. Since the approaches that both systems take are quite similar and there do not exist conflicts in the ways attributes in the user models affect the generation process, we believe that these two approaches can be combined to build a more powerful system.

8.2 Sarner's Model

Sarner ([Sarner and Carberry, 1990]) presented a computational strategy for generating definitions tailored to the user's needs in a task-oriented dialogue. A system that adopts this strategy will provide the user with a definition of an object that is most appropriate for the *task* the user intends to perform in the dialogue.

8.2.1 The Strategy

Sarner's model contains a *knowledge base system* and a *multifaceted user model*. The knowledge base consists of a *generalization hierarchy*, a *plan library*, and a *lexicon*. The plan library contains a set of *domain goals* and *plans* for accomplishing them. The plans are built hierarchically so that plans can be decomposed into subplans to reach any level of detail.

The user model covers the user's *plans and goals*, *domain knowledge* and *receptivity*[1], where the *knowledge* avoids giving the user information he/she already knows and helps choose the terms that are familiar to the user, the *plans and goals* help construct explanations that address the user's needs, and the *receptivity* helps provide information in a form compatible with the user's style of learning.

Sarner argues that the appropriate content of a definition should guide selection of a rhetorical strategy, instead of having the choice of a rhetorical strategy determining content. In order to do so, she adopted a strategy as follows:

1. Weigh all the candidate predicates that might comprise a definition according to the user's receptivity.

2. Evaluate all possible instantiations of each predicate according to its *familiarity* and *relevancy*, which yields the *significance* value for that proposition.

3. Divide the propositions into categories of importance according to the weight of the predicate and the significance value of the proposition.

4. Construct the definition using the more important information, and using the less important information as supporting material.

[1]Note that Sarner uses the term *receptivity* differently from us. Here *receptivity* indicates the *style* of presentation the user prefers, which is very similar to the *preference* attribute in our user model.

USER: I am going to visit Paris. What must I have to pay for my hotel?
SYSTEM: You will need to have a major credit card, travelers checks, or French currency.
USER: Travelers checks?
SYSTEM: *Travelers checks are money instruments which you purchase at a bank*.

Figure 8.4: Definition of Travelers Checks 1

USER: I am going to visit Paris. I am afraid of carrying a lot of cash.
SYSTEM: You can carry a major credit card or travelers checks.
USER: Travelers checks?
SYSTEM: *Carrying travelers checks lets you have convertible funds in a safe form*.

Figure 8.5: Definition of Travelers Checks 2

Two examples are shown in figure 8.4 and 8.5 to illustrate how the definition of travelers checks can vary depending on the user's knowledge and past discourse (which implies the user's plans and goals). The two responses are different because of the evaluation of usefulness to the user in different circumstances determined by previous discourse (for details, see [Sarner and Carberry, 1990]).

8.2.2 The User Model

As discussed earlier, there are three components in Sarner's user model. The first component, the user's *domain knowledge*, is a copy of the system's generalization hierarchy and plans in the plan library with each node and link weighted either 1 or 0, indicating whether it is known to the user or not, respectively.

The second component records the user's *task-related plans and goals* and *focus of attention* from the system's point of view. The plans and goals are represented in a *context tree* which is construct by the TRACK system ([Carberry, 1988]) as the dialogue progresses. The focus of attention and the most recently considered subgoal are also marked in the context tree.

The third part of the user model indicates the user's *receptivity* to various predicates, which plays the same role as the *preference* attribute in our model. Several clues can be used to obtain information on this issue. It can depend on the type of predicates the user responds favourably to, or the type of predicates the user uses.

8.2.3 Comparisons

Sarner's model has a much narrower scope than ours. She looks at *definitional* questions only, and provides *single-sentence* responses, which eliminate the use of *schemata* to organize sentences into paragraphs. Generating single-sentence responses is easier than generating multi-sentence ones since there will not be problems such as organizing the sentences, keeping the discourse coherent, etc.

The user's knowledge is also explicitly recorded in Sarner's model. But in contrast to our approach, it is a copy of the entire knowledge base with each node and link marked with a belief factor of either 0 or 1. This is inefficient when the system has a large knowledge base and a large number of users, especially when many users have only a small portion of the domain knowledge (in which case a considerable amount of space is saved in our model, since we record only the part of the knowledge base that the user knows about).

The basic idea for choosing the appropriate proposition in Sarner's model is similar to ours. Both the candidate predicates and their instantiations are evaluated, but the approaches adopted are not the same. Sarner first chooses among the predicates depending on the user's receptivity, then considers all the possible instantiations of the chosen predicate and chooses the most suitable one. Our approach is to instantiate all the predicates first, choose the best one for each predicate and then evaluate the predicates with their most appropriate instantiations. We evaluated the predicates and the propositions in a different order because we claim that the way a predicate is instantiated influences its chance of being chosen as the most suitable one.

In addition to the evaluation order, the weighting schemes differ considerably. In Sarner's model, the predicates are weighed depending on the user's receptivity, and the propositions are evaluated according to a familiarity value and relevance value. The proposition having the highest combined weight will be chosen as the next utterance.

The *familiarity value* used here is similar to the attribute *user's knowledge* in our model, indicating how familiar the user is with a particular concept. The *relevance value* has the same idea as our *focus of attention*, which also measures how relevant a concept is to the current focus.

Sarner uses the equations

$$f = \frac{e^{6b(2-b)} - 1}{e^6 - 1}$$

and

$$r = e^{-(\frac{d}{4})^2}$$

to calculate the familiarity and relevance values, where *f* is the familiarity rating, *b* the belief factor, *r* the relevance rating and *d* the number of focus shifts.

We believe that although these equations serve to show the relationships between certain factors, and may be capable of capturing the features of the familiarity and relevance values, the approach is somewhat unclear. It is easier to see what the familiarity and relevance values would be, with a method like the one we use, where there are separate weighting functions for every factor that influences the generation process, and where these influences are explained more clearly.

8.3 Moore's PEA System

Moore ([1989], [Moore and Swartout, 1989]) investigated a different area in natural language generation, in providing *responses* and *explanations* to advice-seekers. Her work emphasizes *clarifying misunderstood explanations*, *elaborating on previous explanations*, and *responding to follow up questions*, which other previous explanation systems have never dealt with.

8.3.1 System Overview

In order to provide a reactive approach to explanation, Moore built a Program Enhancement Advisor (PEA) ([Neches *et al.*, 1985]) using the Explainable Expert Systems (EES) framework ([Neches *et al.*, 1985]) as a testbed for her work on explanation generation. PEA is an advice-giving system intended to help users improve their Lisp program styles.

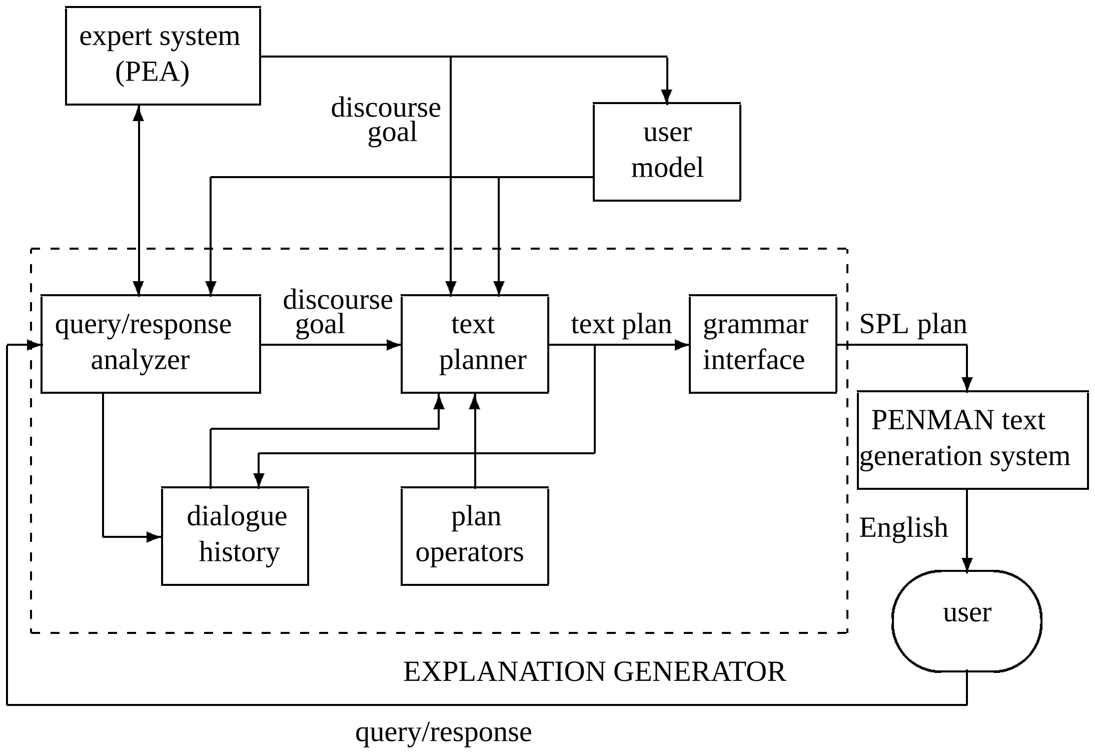


Figure 8.6: The PEA System Architecture

The architecture of the explanation generation facility and its relation to the PEA expert system is shown in figure 8.6.

As we can see from the system architecture, the user's question or response is first analyzed to form a *discourse goal*. The text planner then consults the expert system and the user model to create a text plan which is passed on to a grammar interface to produce input form for the PENMAN ([Mann and Matthiessen, 1983]) text generation system. PENMAN then generates the desired English text as response from the expert system.

The text-planner uses a top-down hierarchical expansion planning mechanism. In order to satisfy a discourse goal, the planner searches in the *strategy library* for strategies that can achieve the goal. The strategy is further expanded into *subgoals* which have to be achieved separately. Throughout the planning process, the system keeps track of all the assumptions made about the user's knowledge and the alternative plans that can be chosen. When the user indicates that explanation or clarification is needed, the dialogue history is examined, so that either alternative plans can be provided or knowledge that was previously assumed to be known to the user will be further illustrated.

A sample dialogue with PEA is shown in figure 8.7. In this example, most of the follow up questions are ambiguous, such as *why* and *huh*. In these cases, the system has to disambiguate them by first finding out all the candidate interpretations and choosing amongst them. The criteria for resolving the ambiguity include past discourse, user's goal, and so forth.

8.3.2 The User Model

PEA includes a user model, which itself includes information about the *user's goals*, the *concepts the user is familiar with*, and *facts the user believes*, from the system's point of view.

Moore models the user's knowledge in a similar way as we do, by assuming that the user's knowledge is a *subset* of the system's knowledge. PEA acquires information in the user model from a variety of sources. Some basic concepts are assumed to be known by all the users, and the top-level goal of *enhancing a program* is assumed to be true for every user. Another rich source of information comes from the input program the user provides to enhance. PEA analyzes the program and finds out what Lisp structures the user knows. Based on its own knowledge about the Lisp structures and the information previously obtained, PEA uses its inference rules to infer what other properties the user might know. However, the inference rules employed in PEA are too simplistic, and may cause the system to attribute too much or too little to the user's knowledge state. Moore claims that, since PEA has the ability to answer follow up questions and give further explanation, SYSTEM : What characteristics of the program would you like to enhance?

USER : Maintainability.

.

.

.

SYSTEM : You should replace *(setq x 1)* with *(setf x 1)*.

USER : Why?

SYSTEM : I'm trying to enhance the maintainability of the program by applying transformations that enhance maintainability. *Setq-to-setf* is a transformation that enhances maintainability.

USER : Why?

SYSTEM : Transformations that enhance maintainability are defined as transformations whose right-hand-side's use is more general than its left-hand-side's use. *Setf* can be used to assign a value to any generalize-variable. *Setq* can only be used to assign a value to a simple-variable.

USER : What is a generalized-variable?

SYSTEM : Simple-variables are storage locations named by symbols. This concept can be generalized to a storage location that can be named by any accessor function. These storage locations are called generalized-variables.

USER : Huh?

SYSTEM : For example, the *car* and *cdr* of a cons are generalized-variables, named by the the accessor functions *car* and *cdr*. Other examples are an element of an array or a component of a structure.

Figure 8.7: Sample Dialogue with PEA

these errors are not critical.

Although PEA employs a user model, Moore argues that requiring a detailed user model is unrealistic. Therefore, PEA is designed to be not critically dependent on information in the user model. Instead as much information as possible is obtained from the input program, the question, and the dialogue history. It is still able to operate properly even if no extra information is known about the user or the information does not correctly reflect the user's knowledge.

8.3.3 Comparisons

Moore's work looks at a considerably different aspect of natural language generation. She emphasizes providing explanations and disambiguating follow up questions which are not discussed in our work.

Her use of the user model is also very different from ours in the sense that we consider information in the user model to be a very important source of deciding *what to say* in the response. Moore, on the other hand, does not consult the user model to give an appropriate answer, but allows PEA to provide a more general first-shot answer and expects the user to ask follow up questions if he/she does not understand it. We believe that a user will be more pleased if he/she can understand the first answer the system provides. Consulting the user model for the user's domain knowledge prevents the system from giving information that is either too difficult or too obvious for the user. In the example given in figure 8.7, the term *generalized-variable* would not be used if a user model indicating that the user does not know this term is consulted beforehand.

The discourse history is not explicitly recorded in our model, but we use a different approach with a similar effect. In our model, all the nodes and links in the relevant knowledge pool corresponding to information in the proposition are tagged after each utterance.

In selecting later utterances, we have an indication of which part of the relevant knowledge pool has been mentioned in this session so that it will not be chosen again unless the user asks for further illustration. Our method lacks the *ordering* of the information provided. In other words, Moore's approach can be thought of as being implemented with a *stack* and ours using a *set*. But since we do not emphasize reasoning about the previous discourse to provide explanations, it is sufficient just to keep track of the information in an unordered fashion.

As with TAILOR, PEA lacks the ability to update the user model as the dialogue progresses. Though this is not an essential point in PEA and Moore claims that "when things should migrate from the dialogue history to the user model is an open question", we argue that losing track of the user's newly obtained knowledge greatly decreases the value of maintaining a user model. Therefore we propose a solution of keeping track of the past discourse and record new information into the user model after the conversation. Although this solution is not ideal, it provides the system with the ability to work over multiple sessions with the same user.

8.4 Wolz's GENIE System

Paris ([1987]) showed how the user's level of expertise can be used to tailor the description of complex objects. Wolz ([1990a], [1990b]) extended her work to the description of *tasks* and introduced a richer user model. Her system, GENIE, emphasizes how the user model impacts content planning in a *task-centred* question answering system.

8.4.1 System Overview

GENIE's goal is to provide an answer that responds informatively to the user's question and that opportunistically enriches the user's expertise. It operates in the Berkeley Unix Mail domain, and accepts five types of questions. The questions are all *task-related* in which a *goal*, a *plan*, or both are provided.

In GENIE, the question answering process is divided into three stages, *understanding the user's question*, *selecting a course of action to suggest*, and *formulating the answer in a manner appropriate to the user's background*. In order to accomplish the task, the system is divided into four components:

1. An *observational expertise* that includes the mechanisms for understanding the user's queries.

2. A *domain expertise* that includes knowledge of the functionality of the system.

3. An *analytic expertise* that reasons about the consequences of executing tasks with a particular functionality.

4. An *explanatory expertise* that provides information appropriate for the user, without being too succinct or too verbose.

The five basic question types are all about *how to do tasks*. Associated with each question type, there is a relationship between a computational goal and a plan of action, along with a set of expectations about what the answer will contain. The answer may consist of *introduction*, *reminder*, *distinction clarification*, or *misconception elucidation*, which can be used either to *respond* to the user's question or to *enrich* the user's knowledge. GENIE reasons using past discourse and information in the user model to generate

Question 1 : Will “mail kathy” let me send a message to Kathy?

Situation : User in read mode, new mail exists.

Yes. Also, you can send mail from Unix. Type “mail kathy” at the Unix prompt.

Figure 8.8: Response Generated by GENIE 1

Question 2 : Why doesn’t “mail mckeown” let me send a message to Kathy?

Situation : User in read mode, new mail exists.

Kathy’s email address is not mckeown. Kathy’s email address is kathy.

Figure 8.9: Response Generated by GENIE 2

a single *first-shot* answer and expects follow up questions which are handled in the same way as was done by Moore ([Moore and Swartout, 1989]).

The responses GENIE generates to the questions *will “mail kathy” let me send a message to Kathy* and *why doesn’t “mail mckeown” let me send a message to Kathy* are shown in figures 8.8 and 8.9.

8.4.2 The User Model

In order to provide sufficient information to decide what is the *best* answer for a particular user, GENIE has a three part user model:

1. A *situational context* provides information about the current conversational environment.
2. A *discourse context* provides details about the user's goal in asking the question.
3. A model of the *user's domain knowledge* includes knowledge of *intention* (how to accomplish certain domain tasks) and *causality* (the results of the actions).

Part of the user model is constructed by the system experts discussed earlier. The observational expert analyzes the user's question and constructs the *discourse* and *situational* contexts. The domain expert models the user's domain knowledge from GENIE's point of view, in which the user's *goals*, *plans*, and the *user initiated actions* are included. Other experts consult the user model to construct the most appropriate plan and response. The analytic expert reasons about the relationships between the entities under consideration and provides relevant information for the explanatory expert, which filters the information to generate an answer that meets the user's needs.

8.4.3 Comparisons

Wolz's work is more similar to Moore's than to ours in the sense that they both provide first-shot information and accept follow up questions for explanation or further illustration.

In GENIE, the user model consists of three parts, the situational context, the discourse context, and the user's domain knowledge. We argue that the user model should contain long-term information; therefore, it should not include information about the conversational environment and the topic. However, the situational context in GENIE contains information about the current interactive environment and the discourse context includes information about the on-going dialogue; hence, we maintain that they should not be recorded in the user model. In our model, we do not take the conversational setting into account, but the *previous discourse*, which is also specific to the interaction, is kept within the relevant knowledge pool and discarded after the conversation, instead of being recorded in the user model.

Wolz claims that the explanatory expert in GENIE performs an important task, *filtering* or *revision*, which is not done in other systems using *schemata* or *Rhetorical Structure Theory (RST)* ([Mann and Thompson, 1987]) based planning. In the generation process, the potential answer is formed by the analytic expert, and the explanatory expert consults the user's domain knowledge in the user model to decide which part of the answer should be included, and which should be excluded. In our model, we use a different strategy by consulting the user model before forming the potential answer. In this way we aim to select information appropriate for that particular user, therefore saving the effort of revising the answer.

As shown in figure 8.8, in addition to the response the user expects, extra information is provided for sending mail messages in the Unix environment. This idea of opportunistically enriching the user's expertise is quite similar to our approach of choosing more than one schema in an answer. In GENIE, there is no information in the user model to indicate whether the user is willing to learn more or not; the system provides extra information whenever it is possible. We argue that additional information should be provided only if the user has the ability to learn more and remember it. Therefore, in our model, the user's receptivity and memory constraints are recorded to help decide if this extra information should be included or not.

8.5 McCoy's ROMPER System

McCoy ([1985],[1986],[1988]) dealt with the problem of the *user's misconceptions*[2]. Her goal was to generate responses similar to the responses of human conversational partners to correct the user's misconceptions by reasoning on a highlighted model of the user to identify possible sources of the error.

8.5.1 System Overview

Through a transcript study, McCoy discovered that a response to a misconception usually contains three parts: a *denial of the incorrect information*, a *statement of denial and correction given*, and a *justification for the misconception*. In order to find out the cause of the misconception, she introduced a new notion of *object perspective*, which acts to filter the user model, highlighting those aspects which are made important by previous exchanges. This perspective helps to reason for possible support for the misconception.

McCoy implemented a system called ROMPER (Responding to Object-related Misconceptions using PERspective) to deal with misconceptions in a financial securities domain. ROMPER takes a specification of the *information that is inconsistent with the system's model of the world*, the *current perspective*, and a *record of past focus* as input, and by consulting its knowledge and perspective library, it generates a response specification which is passed on to McDonald's MUMBLE system ([McDonald, 1986]) to produce the actual English output. The ROMPER system architecture is presented in figure 8.10.

As shown in the system diagram, two types of misconceptions are handled in ROMPER, *misclassification* (classifying an object wrongly) and *misattribution* (giving an object an attribute it does not have). Associated with each kind of misconception, there are

[2]Here, misconception is defined as the discrepancy between system beliefs and user beliefs.

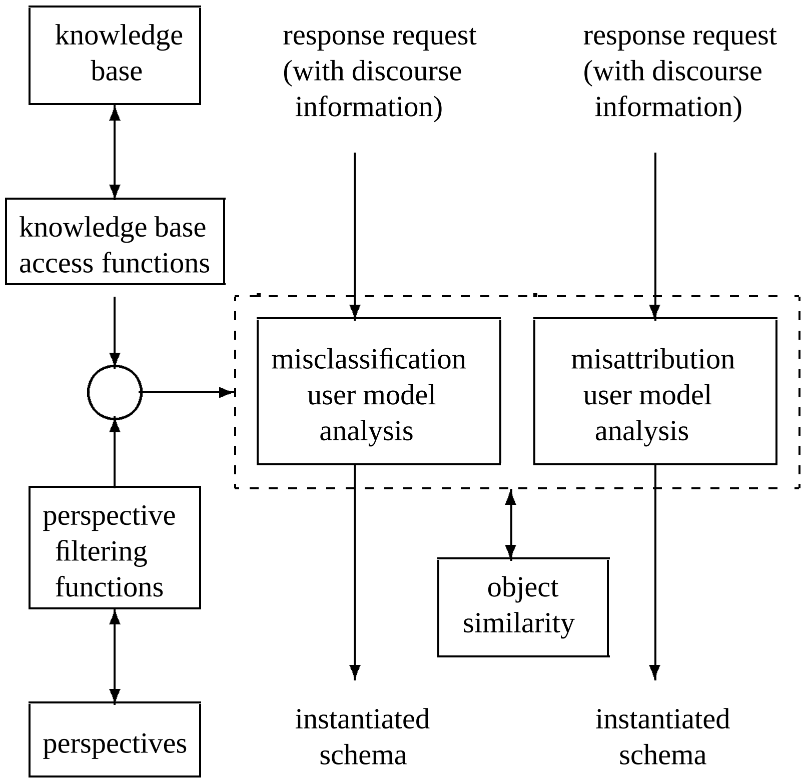

knowledge base
response request (with discourse information)
response request (with discourse information)
knowledge base access functions
misclassification user model analysis
misattribution user model analysis
perspective filtering functions
object similarity
perspectives
instantiated schema
instantiated schema

Figure 8.10: The ROMPER System Architecture

U : I am interested in investing in some securities to use as savings instruments.
I want something short-term and I don't have a lot of money to invest so the instrument must have small denominations. I am a bit concerned about the penalties for early withdrawal. What is the penalty on a T-Bill?
R : T-Bills don't have a penalty. Were you thinking of Money Market Certificates?

Figure 8.11: Sample Output of ROMPER 1

U : I am interested in investing in some securities. Safety is very important to me, so I would probably like to get something from the government. I am a bit concerned about the penalties for early withdrawal. What is the penalty on a T-Bill?
R : T-Bills don't have a penalty. Were you thinking of T-Bonds?

Figure 8.12: Sample Output of ROMPER 2

a few response strategies, represented as response schemata. Depending on the current perspective taken as input by ROMPER, it calculates the *similarity values* between the object in focus (the one mentioned in the question) and all the possible objects for causing the misconception (for details, see [McCoy, 1988]). The object with the highest similarity value is assumed to have caused the confusion and a proper response schemata will be chosen to generate an explanation for clarification.

In figures 8.11 and 8.12 we show how ROMPER acts differently to the same user having the same misconception under different contextual situations.

8.5.2 The User Model

The ROMPER system does not actually do its reasoning on a separate user model. Instead, it performs a *user model analysis* which is actually done on the system representation. McCoy contends that, when the user exhibits a misconception and the system does not have a good model of what the user knows, it is reasonable for the system to do reasoning on its own knowledge base and attribute the knowledge involved to the user. In other words, ROMPER uses the processing done for correcting misconceptions to help build the user model.

As mentioned earlier, there are two types of user model analyses used by ROMPER, the *misclassification user model analysis* and the *misattribution user model analysis*. The misclassification user model analysis decides which correct schema to follow when a misclassification occurs. Three schemata can be chosen, *like-super*, *like-some-super*, and *no support*. *Like-super* is used when the user model shows that a possible reason for the misconception is because the user classified the object as a superordinate to which it does not belong. *Like-some-super* is chosen when the user thinks that the object is similar to a superordinate's subtype. Finally, if no reason can be found for the misconception, the system simply chooses *no support* to inform the user with his/her misconception without an explanation.

The misattribution user model analysis, on the other hand, decides which schema to choose when a misattribution arises. Again, there are three candidate schemata, *wrong object*, *wrong attribute*, and *no support*. The *wrong object* schema is chosen when the user wrongly attributed a property to an object and the *wrong attribute* schema is used when there is an evidence that the user has confused an attribute with another similar attribute of the same object.

Thus, by building a *dynamic user model* while making an analysis on the system's own knowledge base, ROMPER is able to find out the possible causes of the misconception and provide the user with a correction and an explanation.

8.5.3 Comparisons

McCoy looked at a very different area of user modeling from ours, concentrating on dealing with the user's *misconceptions*. In our system, we only handle the cases where the user has *incomplete information*. In these cases the user is not aware of the existence of some intermediate level objects; therefore, he/she has a different hierarchical structure from the system's knowledge base. Though both systems can be used as natural language interfaces to a database system or an expert system, we believe that ROMPER is more suitable to serve as a subsystem that is evoked by, say, a question answering system when a user's misconception is detected during the system's interaction with the user. When a misconception arises, the main system passes the information to ROMPER, including the detected misconception, past discourse (which is usually recorded in most recent natural language interface systems), and the current perspective (this might have to be added to the main system, or be inferred by ROMPER. For a discussion of how to choose an active perspective, see [McCoy, 1988]). Thus, ROMPER corrects the user's misconception and returns control to the main system from which the user continues the interrupted conversation with the system.

An important issue in ROMPER is to decide the *similarity* between objects. Most existing systems determine object similarities depending on the *distance* between the two objects in the hierarchical structure, which is the approach adopted in our model. McCoy claims that the similarity between two objects is highly dependent on the perspective, especially when an object has more than one parent. Therefore she uses the current perspective to highlight the attributes that should be considered, and employs Tversky's metric ([Tversky, 1977]) to calculate the similarity values. We agree with McCoy that two objects may be considered similar under some circumstances, but different in other cases. Since we do not choose to work in a domain where there exists many views to an object, perspective is not as critical in our model as in ROMPER. We believe that simply calculating the distance between two objects provides us with sufficient information needed in our model. But we do realize that to apply our model in a more complicated domain with different *views* on an object, a similarity-calculating strategy similar to ROMPER's should be developed.

The current *perspective* is McCoy's method of capturing the user's view of the knowledge base. Our method is to store a model of the user's knowledge explicitly, and to project this information into the relevant knowledge pool constructed for generation. McCoy's perspective allows the user's knowledge to be represented as something other than a simple subset of the system's knowledge (as in our model). In other words, McCoy has a different representation of the user's knowledge, which may be useful for certain users (with misconceptions) or certain domains (where multiple views of the knowledge can be easily categorized).

As mentioned before, there are some schemata associated with each user model analyzer. Here, the schemata are in fact a collection of predicates that might be chosen as output. They do not specify the order in which the propositions should be given. In our approach using the ATNs, the order of the propositions is semi-fixed, which has a disadvantage of lack of flexibility. In the method ROMPER used, more varieties of text can be produced, but also a more powerful sentence organizer must be developed to guarantee the coherence of the generated text.

8.6 Sarantinos and Johnson's EXPLAIN system

Sarantinos and Johnson ([1990], [forthcoming]) examined the nature of explanation dialogues and designed an explanation system to provide appropriate responses and explanations to the questions posed by different users. They also developed a theory of explanation called *Extended Schema based Theory (EST)* to provide powerful and complete question understanding and explanation generation in an efficient computational form.

8.6.1 System Overview

Sarantinos and Johnson extended McKeown's schemata to work in an explanation generation environment. Instead of choosing one schema for a question according to its question type, each question is analyzed depending on the a classification of the user's knowledge (e.g. novice, expert, and partial expert) and *previous discourse* to construct a *question path*. The question path is a list of question types which can be instantiated by different schemata or predicates. The overall structure of their system EXPLAIN is similar to TEXT, as shown in figure 8.13.

EXPLAIN operates in a *facial skin care* domain. We see that the input question is first analyzed and the question path determined. The question path reflects the question's semantics and its relation to previous discourse and is passed on to the *explanation plan selector* which selects appropriate schemata/predicates and generates a final explanation plan. The schemata are then instantiated in a way similar to the process in TEXT except that previous discourse and information about the user are taken into account. Note that the knowledge base and the input question together create a relevant knowledge pool which provides information for matching the predicates. After an explanation is given, EXPLAIN stores the various knowledge elements used in the question and explanation to

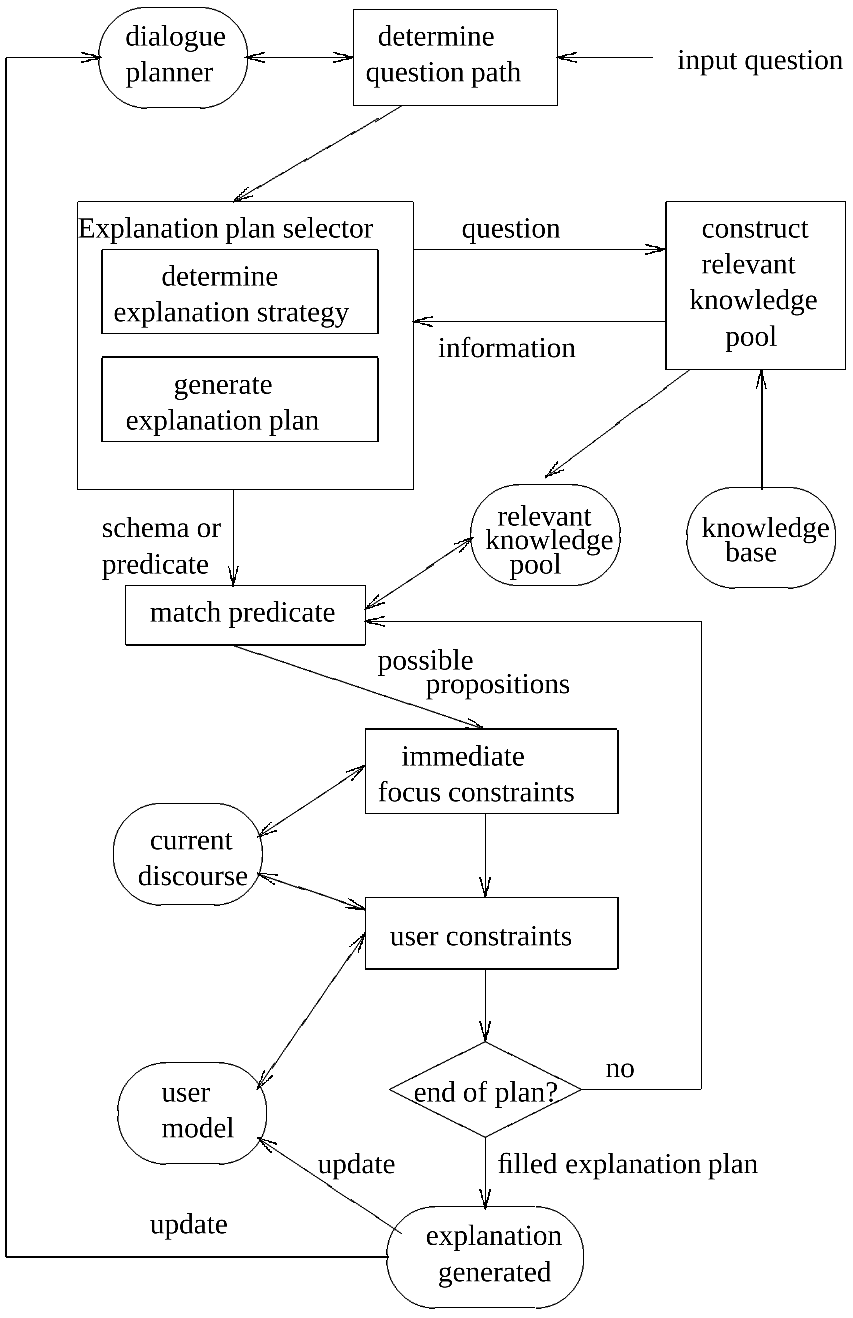


Figure 8.13: The EXPLAIN System Architecture

keep track of how the global focus of the conversation shifts from one topic to another. This process helps make decisions in further question type instantiations.

Through an analysis of natural dialogues between experts, novices, and partial experts, Sarantinos and Johnson identified 28 question types, including *identification*, *evaluation*, *feature analysis*, and so forth. The relations between question types and question paths are studied, and the explanation schemata are associated with each question type.

In figure 8.14 we show an example of the explanation generated by EXPLAIN. The question path and explanation plan inferred by EXPLAIN are shown, followed by the explanation generated.

8.6.2 The User Model

Through empirical observations, Sarantinos and Johnson claim that potential variations exist between novice and partial expert users of an explanation system. In order to tailor the explanations to users of various levels of expertise, they include in their system a user model which keeps track of the user's domain knowledge.

The user model employed in EXPLAIN serves to provide the system with information about the user's underlying knowledge. To acquire knowledge about the user, the implicit knowledge acquisition rules described in [Kass, 1987] are adopted. An example of these rules is the *concept generalization rule* which says: "If the user model indicates that the user knows several concepts that are specializations of a common, more general concept in the domain model, the user modeling module may conclude that the user knows the more general concept, and the subsumption relationship between the more general concept and the more specialized concepts as well".

Some knowledge is assumed to be known to the user, including information mentioned in previous discourse and concepts mentioned in the user's question. Using this

Question : What is a scrub; how do I use it?

The question path is:

identification => concept-completion => frequency, timing, contents, place order

The explanation plan is:

identification predicate

attributive schema

Generated explanation:

The scrub is an exfoliating product whose purpose is to exfoliate the skin without drying. The scrub is usually referred to as facial scrub or exfoliating cream. The scrub contains grains. You apply the scrub with your finger-tips by doing circular movements. How frequently you use it depends on your skin type. For oily skin types you use it twice a week whereas for dry skin types once a week. Use it all over the face avoiding the eye-area. It costs about $2.5 and it is best not to use it if you are not 25 years old or more. The exfoliating scrub is very good for sensitive skins.

Figure 8.14: Sample Explanation Generated by EXPLAIN

information and the knowledge acquisition rules mentioned above, EXPLAIN is able to make inferences as to which entities are known to the user and which are not. The list is then updated dynamically as the dialogue progresses to prevent repetitions and redundant information.

The primary use of user models for varying the explanation is in the process of selecting the question type intended by the user. Then, focus constraints control the actual generation. The user's knowledge is also mentioned as an influence in the selection process during generation, but how to employ that user knowledge to tailor the output is not explicitly stated and no general algorithm is provided.

8.6.3 Comparisons

The EXPLAIN system, though also based on the *schema based theory* used in McKeown's TEXT system, looks into aspects considerably different from ours. Sarantinos and Johnson emphasize *understanding the question*, *providing explanations* and *handling follow-up questions*, while we stress the generation phase of a question answering system and study how various aspects in a user model influence this process.

First of all, Sarantinos and Johnson looked into the problem of determining question types more carefully than we did. In our model, we determine the question type using the user's role and the default value associated with each role. In EXPLAIN, a question is interpreted in terms of a path of classified question types which is further processed to select the appropriate explanation plan. The selection of question types depends on the user's domain knowledge and previous discourse, which results in explanations tailored to different users.

The updating mechanism used in EXPLAIN is also different from ours. Inherited from TEXT, both EXPLAIN and our model create a relevant knowledge pool based on the system knowledge base and the input question to restrict the scope of the information in the responses. Furthermore, the two systems both keep track of previous discourse to prevent repetitions within the same conversation. In our model, this is done by tagging the corresponding nodes and links in the relevant knowledge pool after each utterance, while in EXPLAIN, the user model is updated as the dialogue progresses. Our algorithm is designed to improve efficiency of processing. Since information is stored in the relevant knowledge pool, the system does not have to consult the user model every time it comes to a decision point and does not have to update the user model after each utterance. We argue that the interaction between the system and the user is a *real-time* process; therefore, the time spent on updating the user model after each utterance is noticeable to the user, while updating after the conversation is more agreeable to the user.

Although EXPLAIN does not consider many different aspects of the user in the user model, the effort spent in studying the question types and question paths in order to construct a explanation plan is beneficial. Since both EXPLAIN and our model are based on the TEXT system, we believe it is possible to adopt the question analysis phase of EXPLAIN (modified to work in a question answering system) in our model to process a wider range of question types and to be more sensitive to ongoing dialogue.

8.7 Hovy's PAULINE System

Hovy ([1988a],[1988b],[1990]) looked into the question of *how* and *why* people say the same thing differently to different people or even to the same person in different circumstances. He is the first researcher to investigate the issue of building *pragmatic goals* into a text generation system.

8.7.1 System Overview

Unlike the other systems described above as question-answering systems, Hovy's PAULINE (Planning And Uttering Language In Natural Environments) system produces *stylistically* appropriate texts that describe an event according to the various settings that model pragmatic circumstances.

The architecture of PAULINE is depicted in figure 8.15. The entire planning process consists of three stages, namely, *topic collection*, *topic organization*, and *realization*. The input topic[3], is first passed to the topic collection module which collects from the input elements additional representation elements and determines which aspects of them to say. There are three plans available in PAULINE, the *describe* plan which finds descriptive aspects of objects, the *relate* plan which relates events and state-changes, and the *convince* plan that selects topics that will help convince the hearer of some opinion.

Once the plan is chosen, the *topic organization* module performs the task of finding appropriate groupings and interpretations of the candidate topics, finding appropriate ways to juxtapose them in multi-predicate phrases, and finding ways of expressing relationships amongst them. At this stage, the outline of the output text is decided, including the topics to be included, how they are ordered, and so on. This information is further passed on to the *realization* module, which determines the organizations within sentences, and selects appropriate *clauses* and *words*.

In addition to the input topic, PAULINE also takes as input a set of *pragmatic goals*[4]

[3]There are three sets of examples, i.e. three *topics*, that PAULINE can operate on, *the shantytown affair*, *a fictitious presidential election*, and *a model of the behaviour of a judge* (for details, see [Hovy, 1988a]).

[4]There are thirteen pragmatic goals used in PAULINE, including *access knowledge*, *make the hearer like/dislike the speaker*, etc., These pragmatic goals are represented by some interpersonal goals and conversation settings. For more details, see [Hovy, 1988a].

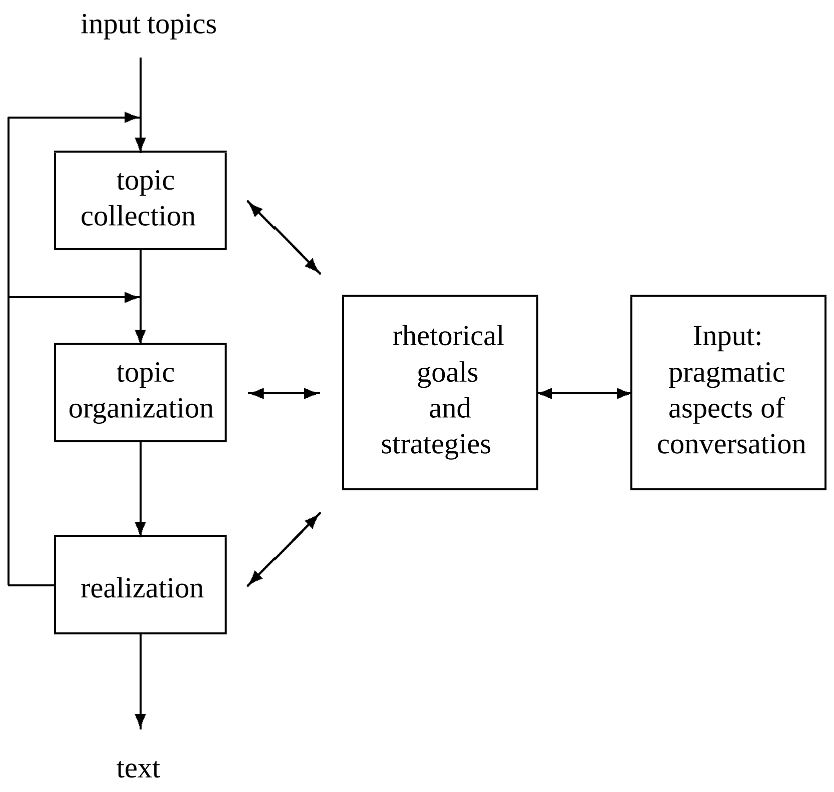

input topics
topic collection
topic organization
realization
text
rhetorical goals and strategies
Input: pragmatic aspects of conversation

Figure 8.15: The PAULINE System Architecture

which activate a number of intermediate *rhetorical goals*[5] that control the *style* and *slant* of text. All three stages in the text planning process interact with the derived rhetorical goals, since the goals influence not only the topics to be included and how they are organized, but also the sentence structures and lexical choices.

Note that the output from the realization component provides feedback for both the topic collection and topic organization components. Hovy contends that when people start speaking, they usually have a vague idea of what topics they will cover but leave the details for later consideration. Therefore, he suggested that content planning should be performed only when necessitated by the realizer.

To show how PAULINE generates different styles of texts depending on different pragmatic goals, two sample outputs describing the shantytown affair, together with their pragmatic goals, are shown in figures 8.16 and 8.17. Note that the first example is intended to be output by a passerby describing the issue, and the second one a newspaper article.

8.7.2 Comparisons

Unlike our use of schemata as discourse strategies inherited from TEXT, Hovy introduced a different approach by using a *limited-commitment planning* paradigm. The most important aspect of limited-commitment planning is that the *planning* (deciding *what to say*) and *realization* (deciding *how to say it*) components are *interleaved*, which allows planning to take into account unexpected syntactic opportunities and inadequacies. However, in our approach, the content is completely decided before the realization component performs its task. Though Hovy's approach guarantees that the realizer will be able to fulfill the planner instruction (if it cannot realize the instruction, the planner will be asked to

[5]Twelve rhetorical goals are used in PAULINE, e.g. *formality*, *simplicity*, *haste*, and *openmindedness*.

- Time : *some*

- Tone of interaction : *informal*

- Speaker's opinions : *neutral*

- Depth of acquaintance : *strangers*

- Goal to affect hearer's opinions : *none*

Yale University punished a number of students for building a shantytown on Beinecke plaza by arresting 76 students and tearing the shantytown down one morning in April. The students wanted Yale to divest from companies doing business in South Africa. Finally, the University gave in and allowed the students to rebuild the shantytown.

Figure 8.16: Sample Output of PAULINE 1

- Time : *much*
- Tone of interaction : *formal*
- Speaker's opinions : *neutral*
- Depth of acquaintance : *strangers*
- Goal to affect hearer's opinions : *none*

In early April, a shantytown – named Winnie Mandela City – was constructed on Beinecke plaza by several students, so that Yale University would divest from companies doing business in South Africa. At 5:30AM on April 14, it was destroyed by officials; also, at that time, the police arrested 76 students. The students requested that Yale give them permission to reassemble the shantytown while several local politicians and faculty members expressed criticism of Yale's action. Finally, the University permitted the students to reconstruct it and, concurrently, Yale University announced that a commission would go to South Africa in July to examine the system of apartheid.

Figure 8.17: Sample Output of PAULINE 2

replan), they also come at a cost. Since our work does not emphasize the style of the response, the feedback from the realization component is not as critical. Therefore, a traditional top-down planning method will be sufficient to serve our needs.

PAULINE does not have a *long-term* user model, but takes the pragmatic goals as input. The pragmatic goals involved in PAULINE can be categorized into *information about the speaker*, *information about the hearer*, *the speaker-hearer relationship*, *the conversational atmosphere*, and *the interpersonal goals between the speaker and the hearer*. Among those categories, the *information about the hearer* will be considered similar to our user model, but the rest are not discussed in our work. There are several reasons for this. In our system, the speaker is not considered a *person with affection* as in PAULINE, but is what we refer to as the system, a *machine simulating an expert*. Therefore, it is supposed to be knowledgable in its domain, to be very patient, and so forth, which give fixed-values to the aspects considered in the information about the speaker. Since the speaker is not considered a person, the *speaker-hearer relationship* and the *interpersonal goals between the speaker and the hearer* do not exist. As for the *conversational atmosphere,* there also do not exist as many variables as in PAULINE (being on the street, in a quiet place, etc.). We assume that the learning environment is quiet and the hearer has lots of time to learn from the system.

PAULINE uses twelve rhetorical goals to realize the pragmatic settings. Some of them are related to the information of the hearer, and are similar to the attributes recorded in our user model. The rhetorical goal *simplicity* indicates the complexity of the sentence structure as well as the phrases and words chosen, while the rhetorical goal *detail* controls the level of detail of topics generated. Both of them are used in a way that is similar to the idea of *stereotyping* the users according to their domain knowledge (since PAULINE only has three values for each goal). The rhetorical goal *openmindedness*, which indicates the user's willingness to consider new topics, represents the same idea as the *receptivity* attribute in our user model.

Hovy also discusses the usage of *reminders* in PAULINE, but it is applied somewhat differently from ours. Hovy's inclusion of reminders is done at the topic collection stage, deciding whether related situations that happened in the past can be used as examples in the text. He does not consider whether the user knows that topic or not, or if he/she has a good or poor memory. Reminding in PAULINE is treated as an issue of *related topic inclusion*.

The idea of PAULINE and our model can be integrated to build a natural language interface to a database system that responds according to the system's and user's aspects and the conversational settings. By extending our user model, we can provide PAULINE with the necessary information about the speaker, the hearer, and their relationships. Also, our user's knowledge base can compensate for PAULINE's deficiency in stereotyping the users according to their expertise in the domain. But because of the different strategies used in the generation process (schemata in our model and limited-commitment planning in PAULINE), some problems in the planning stage remain to be solved. By including more of PAULINE's pragmatic goals in our model, greater variation in determining *how to say it* can be achieved.

Chapter 9

Conclusions

9.1 Summary and Contributions of the Thesis

9.1.1 Summary of Our Work

In this thesis, we developed a multi-dimensional user model with the following content:

- User's role: In our model the role captures the user's possible goals, and is used to interpret the question type (whether it is definitional or procedural).
- User's knowledge: This is the user's view of the knowledge base, used to record what the user knows and does not know. It is taken as a subset of the system's knowledge base, but can contain *incomplete information*.
- User's preferences: The *style* of presentation the user prefers, used to help choose schemata and predicates.
- User's interest: the user's general interest, mostly helpful in predicate instantiation.

- User's receptivity: This indicates how willing the user is to accept *new* information. It contributes to deciding the number of schemata and the amount of information to be provided.
- User's memory constraints: A classification of users with good/poor memory. It also helps decide the number of schemata, but more importantly, whether reminders should be provided.

We also proposed a Telos-like knowledge representation for recording the user's view of the knowledge base. Using this representation, each node is recorded as a *frame*, together with pointers to its parent(s) and child(ren). The hierarchical structure uses property inheritance to record what the user knows generally about a node, and also allows for properties to be attached directly to the node explained to the user.

Given the content and representation of the user model, we studied when and how the attributes in the user model influence McKeown-style natural language generation. The whole generation process is divided into five stages : *determine question type*, *create relevant knowledge pool*, *select schema*, *instantiate predicate*, and *select predicate*. One or more user model attributes help to make decisions at each stage. They can be summarized as follows:

- Determine question type: user's role.
- Create relevant knowledge pool: user's knowledge.
- Select schema: user's receptivity, memory constraints, and preferences.
- Instantiate predicates: user's interests, receptivity, and memory constraints[1].

[1]In fact, the user's knowledge affects this stage and the predicate selection stage indirectly. Since this influence is recorded in the relevant knowledge earlier, we will not repeat it here.

- Select predicate: user's preferences, memory constraints, receptivity, and interests.

Based on the study of these influences, we proposed an algorithm that explicitly describes how they take place at different stages in the generation process. We also realized that, at some stages, there may exist conflicts amongst the choices various attributes suggest, so we used different weighting schemes to solve this problem. Also, in order to prevent the system from repeating itself, the corresponding nodes and links are marked off in the relevant knowledge pool after each utterance, and the user's knowledge base is updated after each conversation.

9.1.2 Analysis of Our User Model

In this section, we will compare the attributes we include in our user model with the contents we proposed in chapter 3, and analyze them in terms of their dimensions (also discussed in chapter 3).

Contents:

1. Goals and plans: This is captured by the attribute *user's role* in our user model. In our cooking domain, the users are classified into two classes, *chef* or *diner*, and there is a default goal for each class.

2. Preferences and attitudes: The user's preferences of style is included in our model. The aspect *attitude* is partly captured by our attribute *interests*, indicating what the user particularly likes.

3. Knowledge and beliefs: We consider the user's *domain knowledge*. The user's *beliefs*, additional short-term knowledge specific to the conversation, are not handled here.

Table 9.1: Dimensions of Attributes in Our User Model

4. Memory constraints: Included in our model.

5. Interests: Explicitly recorded in our model.

6. Receptivity: Also in our user model.

Dimensions:
The dimensions of each attribute included in our user model are shown in table 9.1. For an explanations of the dimensions, refer to section 3.2. From the analysis of our user model we conclude that it is in fact a model which is *single agent*, *single model*, *descriptive*, with some *dynamic*, *short-term* modifiability and some *individual* specialization.

9.1.3 Contributions

In this section we discuss the main contributions of our work, including how it is different from previous work, and why it is important.

- **Richer information in the user model**: Of the six attributes included in our user model, the user's *knowledge* and *goals* have gained much attention in the past.

The user's *preferences* and *interests* were addressed slightly before, but applied differently. The user's *receptivity* was mentioned in PAULINE ([Hovy, 1988a]), but not discussed in detail, while the attribute *memory constraints* is original in our work. We have discussed in previous chapters *why* these attributes are important and have shown that they indeed influence the choice of content in the responses[2].

- **Deciding how to make use of the user's knowledge**: Existing systems have taken a user's knowledge into account to help tailor the output. It has been quite controversial to decide *what to say or not to say, given what the user knows*. Some people argue that we should not say what is in the user's knowledge, therefore using that information to filter out the material the user knows. Others claim that we should also mention something the user knows as an introduction to the new information provided. We maintain that this depends highly on individuals, or more specifically, on their memory constraints. For people who have a good memory, repeating what they already know is unnecessary. For those with a poor memory, a reminder might be expected before new information is provided to make sure that they can fully understand it. For this purpose, we introduce the new attribute *memory constraints* to the user model.

- **A formal representation of the user's view of a knowledge base**: We proposed a representation suitable for all domains with property inheritance. It extends Telos ([Koubarakis *et al.*, 1989]) to record both *definitional* and *procedural* information. Such a representation is important in developing man-machine interfaces, especially in knowledge base studies. With this formal representation, we can easily

[2]Although we included several attributes in our user model, the proposals for tailoring generation based on each of these attributes are not very sophisticated. This thesis aims to investigate the interaction of user model attributes with respect to tailoring natural language generation, and more detailed analysis of each attribute can be done separately to replace the preliminary proposals used in this thesis.

update the relevant knowledge pool to indicate the parts of the relevant knowledge pool which are known to the user, given the question and the user's knowledge. This valued relevant knowledge pool can then be used for the generation process.

- **An algorithm describing the interaction between the user model and the generation process**: This is an extension of McKeown-style generation which covers both *definitional* and *procedural* questions, and tailors the responses to specific users. We clearly specify *where* in the generation process the influences of the user model take place. We discuss at each decision point, *how* various attributes affect the choices and how these decisions change the output. We claim the importance of being able to handle both definitional and procedural questions, as these are both common questions appearing in many domains. The importance of an explicit algorithm is that it clearly shows *when* and *how* the user model interacts with the generation process. Also, when new attributes are added to the user model, only the decision points and, possibly, the weighting schemes in the algorithm would need to change.

- **Combining various aspects in the user model**: Although people have dealt with combining user modeling and natural language generation in the past, they considered only a small number of the user's attributes. In our work, we show how generation could make use of various aspects of the user simultaneously. It is not surprising that when handling these attributes altogether, the situation becomes much more complicated, and in some cases, conflicts amongst the attributes arise. We have simplified this problem by dividing the generation process into several stages and discussing which attributes influence these stages individually. As for conflict resolution, we proposed various weighting schemes for different stages according to the attributes involved and their importance at that particular stage. The advantages of using these weighting schemes is that they are easy to understand, systematic, and easy to modify when new attributes are added or when a particular attribute is emphasized.

- **Updating the user model**: In order to choose the most appropriate response for a user, the system must have a *correct* view of the user, i.e. have correct information in the user model. Therefore, it is very important to keep the information up-to-date by updating the user model. As shown in table 9.1, we consider the *modifiability* of all attributes in our user model, except the user's knowledge, as *static*. Hence, we concentrate on updating the user's knowledge. We choose to tag the new information introduced in the relevant knowledge pool after each utterance and record the changes in the relevant knowledge pool to the user's knowledge base after each conversation. This guarantees that the system has a correct view of the user's domain knowledge at any point, without spending too much time on recording the changes during the conversation. Although our approach of updating the user's knowledge is simple, we argue that it is valuable because it enables the system to work over multiple sessions with the same user. This is an important issue for systems actually interacting with users, but has not been considered adequately in the natural language generation systems to date.

9.2 Limitations and Future Directions

Research in integrating user modeling and natural language generation is still far from being satisfactory. There are many open questions, but we only a few in this thesis. In this section, we discuss some questions that, based on our solution, look promising for future research.

- **Extend the user's receptivity and memory constraints to include a wider spectrum of values**: In our solution, we stereotype the user's receptivity and memory constraints into two classes each, being either *high* or *low*. In fact, this is a superficial classification. In order to capture these two aspects of the users more precisely, a different measurement should be developed to cover a wider spectrum.

- **Include a plan recognition mechanism**: Again, we stereotyped the users according to their roles, with a default goal for each class. To infer the user's goal more accurately, a plan recognition mechanism should be included.

- **Include Hovy's rhetorical goals and conversational settings**: In Hovy's setting, both the speaker and hearer, their relationship, and the conversational environment are taken into account, while in ours, only the hearer is considered. An enhancement to our solution would be to include Hovy's rhetorical goals in the user model, so that the response is tailored not only to the user but also to the speaker and the particular conversation. The conversational setting can be thought of as part of the information in the user's *beliefs*, which is the user's *short-term* knowledge for a specific interaction. This might sound unnecessary when the speaker is regarded as a *system*, rather than a *real person* (as in our case), but will be helpful when the system is applied to simulate conversation between two persons.

- **Look into how user modeling influences other levels of generation**: As mentioned earlier, this thesis emphasizes the influences of user modeling on the *content* of the generated text. There are other levels in the generation process that can be also be influenced, such as syntactic structures and lexical choices. Bateman and Paris ([1989]) investigated how the user's knowledge influences these stages, and we believe it will be worthwhile to see how the other attributes affect them.

- **Infer the user model**: In this thesis, we do not deal with *how the user model is obtained*. We have an *interview* with every new user to find out the necessary information in the user model. An ideal situation would be one where the system has the ability to infer all the information itself so that the user will not even notice the existence of the user model. This requires much more research on knowledge acquisition and might be very difficult, however.

- **Apply the same principles of the algorithm to other generation methodologies**: In our solution, we focus on McKeown-style generation, which uses *schemata* as discourse strategies. However, there are many other strategies used in various generation systems, and it will be interesting to see if the principles used in our solution can be applied to other generation methodologies.

- **Decide the cut-off point of the relevant knowledge pool**: In chapter 5, we discussed the scope of the relevant knowledge pool for a definitional question. When choosing the focus's ancestors and children, it is difficult to decide how far to go. When the ancestor becomes too general or the child too specific, it becomes less relevant. We believe that it is worthwhile to develop a general algorithm to decide when to cut off the chains in various domains since the size of the relevant knowledge pool directly influences the response time of the system. This is generally related to the question of the most appropriate level of detail to present to a user.

- **Increase the range of possible questions to address**: In McKeown's TEXT system, three types of definitional questions can be asked, namely, *requests for definitions*, *requests for available information*, and *requests about the difference between objects*. In our model, the definitional questions are restricted to the first type, asking about the definition of a certain object. In TEXT, the requests for available information are handled by either the *constituency* schema or the *attributive* schema and requests about the difference between objects are answered using the *contrastive* schema. The constituency schema is used in our model for the definitional questions and the contrastive schema used in TEXT is similar to the combination of the *identification* schema and *contrastive* schema in our model. The only schema left is the *attributive* schema which uses predicates similar to the ones used in the *identification* schema and *constituency* schema. Therefore, we believe that the attributive schema can be easily included in our model to cover all three types of definitional questions addressed in the TEXT system.

Appendix A

The Schemata

The constituency, attributive, and contrastive schemata used in McKeown's TEXT system are listed below.

Constituency Schema

{Attributive / Identification}
Constituency
(Identification^{+}
  Evidence) / Attributive
{Attributive / Evidence}*
{Attributive / Analogy}

Attributive Schema

Attributive
Amplification
Particular-Illustration*
{Classification / Attributive}
Analogy
{Explanation}

Contrastive Schema

{Identification Schema}
(Identification Schema
  Identification Schema) /
(Attributive Schema
  Attributive Schema) /
Constituency Schema
Inference

Appendix B

Tracing the Algorithm in Detail

In this appendix we show the details of deriving the output of example 1 in chapter 7.

The user has user model as shown in figure 7.1, and asks the question *what is a meringue pie*. The algorithm is traced through, and the details are shown in the following.

1. Determine question type: first of all we check the user model for the user's role, which is a diner. From the rules given in section 6.2, we find that for a *what* question, if the user is a diner, it is considered as a *definitional* question.

2. Determine relevant knowledge pool:

   2.1 Determine global focus: the global focus is the object being asked about in the question, which in this case is *meringue pie*.

   2.2 Determine relevant knowledge: according to the discussion given in section 6.3, the relevant knowledge for a definitional question is defined to be the global focus, its descendants to the subplan level, its ancestors to the second level from the top, its siblings and the siblings' children. In this case it includes,

global focus	:	meringue pie	
descendants	:	lemon pie —	make flaky crust
			make lemon filling
			make meringue
			combine meringue pie
		chocolate pie —	make Graham wafer crust
			make chocolate filling
			make meringue
			combine meringue pie
ancestors	:	pie, pastry, dessert	
siblings	:	fruit pie	
siblings' children	:	apple pie, cherry pie	

2.3 Assign importance values: by consulting the user's knowledge base in the user model, assign 1's to all the nodes and arcs in the relevant knowledge pool that are known to the user and 0's to the ones unknown. The valued relevant knowledge pool is shown in figure 7.2.

3. Select schemata:

3.1 Determine candidate schemata: according the pre-determined schemata for definitional questions in figure 4.2, the candidates are *identification* schema, *constituency* schema, and *contrastive* schema.

3.2 Determine number of schemata: since the user has both high receptivity and good memory, X = 2.

3.3 Choose X schemata: since the global focus, *meringue pie*, falls on one of the pre-determined levels, we need the user's preference to help decide which schema to choose. In this case, since the user prefers *attributes*, the *identifi-*

cation schema is chosen. For the contrastive schema, we check in the relevant knowledge pool and find that *apple pie*, a child of *fruit pie* which is a sibling of meringue pie, is known to the user. Therefore, the contrastive schema is included as well.

4. Select propositions:

4.1 Instantiate predicates:

4.1.1 Determine candidate predicates: we are now at the state *ID/* (see figure 6.1), and the only outgoing arc is the *identification* predicate.

4.1.2 Instantiate candidate predicates: according the rule for instantiating an identification predicate in section 6.5, the object will be introduced as the closest ancestor known to the user. Searching upwards in the hierarchical structure, we find that the parent of meringue pie, *pie*, is known to the user. Therefore, the predicate is instantiated as *meringue pie is a kind of pie*.

4.2 Select predicate: since the identification predicate is the only one applicable at this point, it is chosen as the first proposition.

4.3 Update relevant knowledge pool: after the first proposition is provided to the user, the node *meringue pie* and the link *pie – meringue pie* will be marked as known.

4.1 Instantiate predicates:

4.1.1 Determine candidate predicates: we are now at the state *ID/ID*, and the outgoing arcs are the *attributive* predicate, *constituency* predicate, and *example* predicate (by taking the *jump* arc to *ID/DS*).

4.1.2 Instantiate candidate predicates: the predicates will be instantiated and evaluated as follows[1]. Note that the *example* predicate is instantiated to illustrate *chocolate pie*, and is expressed using the *inference* predicate which gives the similarities and differences between chocolate pie and lemon pie.

1. Attributive

It has meringue on top

W(fc) = 1	focus = meringue pie
W(pre) = 1	user likes attributes
W(int) = 0	
W(rec) = 1	user receptivity high, and leads back to the same state
W(mc) = 1	user memory high, and attribute unknown

W(Att) = sum of all above = 4

2. Constituency

There are two instances of meringue pie, lemon pie and chocolate pie.

W(fc) = 1	focus = meringue pie
W(pre) = 0	
W(int) = 0.5	user likes chocolate
W(rec) = 1	user receptivity high, and leads back to the same state
W(mc) = 0.5	nodes known (lemon and chocolate pies), but

[1]The evaluation process should be done in step 4.2. Here we list both the instantiation and the evaluation in the same figure for convenience.

arcs unknown (the relationship between meringue pie and lemon and chocolate pies)

W(Con) = sum of all above = 3

3. Example
Chocolate pie has the same meringue as lemon pie, but has Graham wafer crust and chocolate filling

W(fc) = 0	focus = chocolate pie
W(pre) = 0	
W(int) = 0.5	user likes chocolate
W(rec) = 0	user receptivity high, but leads to a new state
W(mc) = 1	user memory high, and information unknown

W(Ex) = sum of all above = 1.5

4.2 Select predicate: from the weights assigned above, we choose the attributive predicate which is instantiated as *it has meringue on top*.

4.3 Update relevant knowledge pool: this proposition causes the attribute *parts: + top meringue* to be marked as known.

4.1 Instantiate predicates: since we are in the same state, *ID/ID*, the candidate predicates and their instantiations are the same as shown before except that there is no information for the attributive predicate.

4.2 Select predicate: between the constituency predicate and the example predicate, we choose the former, being *there are two instances of meringue pie, lemon pie and chocolate pie*.

4.3 Update relevant knowledge pool: in this sentence, two nodes and two links are introduced. The nodes *lemon pie* and *chocolate pie*, and the links *meringue pie – lemon pie* and *meringue pie – chocolate pie* will be marked as known.

4.1 Instantiate predicates: still, we are at the state *ID/ID*. Now, the only choice is the *jump* arc which leads us to the state *ID/DS*, where the only outgoing arc is the *example* predicate.

4.2 Select predicate: the unique choice for the next proposition is *chocolate pie has the same meringue as lemon pie but has Graham wafer crust and chocolate filling*.

4.3 Update relevant knowledge pool: this proposition introduces the subplans *make Graham wafer crust*, *make chocolate filling* and *make meringue*, as well as the links between these subplans and *chocolate pie*.

4.1 Instantiate predicates: now, being at the state *ID/EX*, the two possible choices are the *example* predicate and *pop*. The example predicate will be instantiated and evaluated as,

Example

Lemon pie has flaky crust, lemon filling, and meringue.

W(fc) = 0	focus = lemon pie
W(pre) = 0	
W(int) = 0	
W(rec) = 1	user receptivity high, and leads back to the same state
W(mc) = 0	user memory high, and information known

W(Ex) = sum of all above = 1

4.2 Select predicate: with W(Ex) = 1 and W(Pop) = 1.5 (default value), the pop arc is chosen, therefore, ending the identification schema.

4.1 Instantiate predicates: since the last proposition brought the identification schema to an end, we are now at the beginning state of the contrastive schema, *C&C/*. The only choice here is the *identification* predicate, and is instantiated to identify the new object as, *fruit pie is also a kind of pie*.

4.2 Select predicate: the only predicate mentioned above is chosen.

4.3 Update relevant knowledge pool: the new focus *fruit pie* and the link *pie – fruit pie* are marked as known.

4.1 Instantiate predicates: the two choices here at the state *C&C/ID* are both schemata, the *identification* schema and the *constituency* schema. Hence, no instantiation is required.

4.2 Select predicates: since *fruit pie* is at the same level as *meringue pie*, the decision process is the same as in step 3. Therefore, the *identification* schema is chosen. Here the identification schema is called again, with the global focus being *fruit pie*. The process will be similar to the one presented above for global focus *meringue pie*, and will not be repeated. The output generated for introducing fruit pie is "*It has top-lattice on top. Apple pie is an instance of fruit pie*".

4.3 Update relevant knowledge pool: the above two sentences introduce the attribute *parts: + top-lattice*, the node *apple pie* and the link *fruit pie – apple pie*.

4.1 Instantiate predicates: after traversing the identification schema, we are at the state *C&C/DS*, with the only outgoing arc the *inference* predicate. This is instantiated by searching for the similarities between the two objects in their parent node and the dissimilarities in the corresponding slots in individual nodes. The inference predicate will be instantiated as *meringue pie and fruit pie both have crusts and fillings, but have different tops*.

4.2 Select predicate: again, the only inference predicate is chosen.

4.1 Instantiate predicates: now at the state *C&C/INF*, the only choice is the *pop* arc, which does not require any instantiation process.

4.2 Select predicate: the pop arc is chosen, and completes the answer to the question *what is a meringue pie*.

5. Update the user model: summarizing all the changes made in the relevant knowledge pool discussed above, we get the overall updated relevant knowledge pool as shown in figure 5.7. This has to be integrated with the original user's knowledge base in the user model to indicate the up-to-date user's knowledge, which is shown in figure B.1.

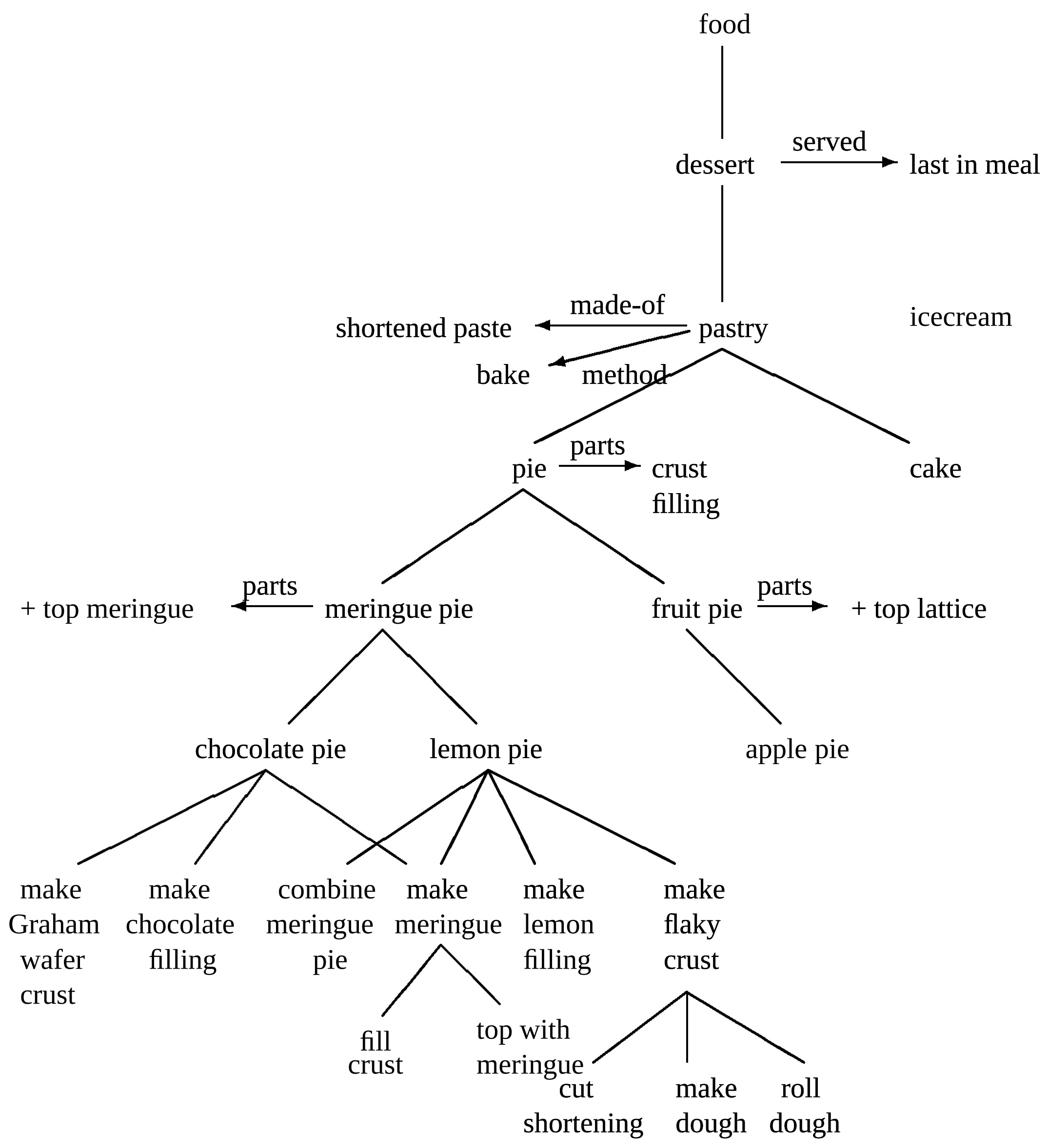


Figure B.1: Updated User’s Knowledge Base

Bibliography

[Allen, 1987] James Allen. *Natural Language Understanding*. The Benjamin/ Cummings Publishing Company, Inc., Menlo Park, California, 1987.

[Appelt, 1985] Douglas E. Appelt. *Planning English Sentences*. Cambridge University Press, New York, 1985.

[Bateman and Paris, 1989] John A. Bateman and Cecile L. Paris. Phrasing a text in terms the user can understand. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1511–1517, Detroit, Michigan, August 1989.

[Britannica Junior Encyclopedia, 1963] Britannica Junior Encyclopedia. Encyclopedia Britannica Inc., William Benton Publisher, 1963.

[Carberry, 1983] Sandra Carberry. Tracking user goals in an information-seeking environment. In *Proceedings of the National Conference on Artificial Intelligence*, pages 59–63, Washington, D.C., August 1983.

[Carberry, 1988] Sandra Carberry. Modeling the user's plans and goals. *Computational Linguistics*, 14(3):23–37, 1988.

[Cohen and Jones, 1989] Robin Cohen and Marlene Jones. Incorporating user models into expert systems for educational diagnosis. In Alfred Kobsa and Wolfgang Wahlster, editors, *User Models in Dialog Systems*, chapter 11, pages 313–333. Springer-Verlag, Berlin, 1989.

[Cohen *et al.*, 1989] Robin Cohen, Marlene Jones, Amar Sanmugasunderam, Bruce Spencer, and Lisa Dent. Providing responses specific to a user's goals and background. *International Journal of Expert Systems*, 2(2):135–162, 1989.

[Computational Linguistics, 1988] Computational Linguistics. Special issue on user modeling. 14(3), 1988.

[Corbett, 1990] Edward P.J. Corbett. *Classical Rhetoric for the Modern Student*. Oxford University Press, New York, 1990.

[Dent *et al.*, 1987] Lisa Dent, Amar Sanmugasumderam, and Bruce Spencer. Using user models for explanation: Introducing thums. Technical Report CS-87-38, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, June 1987.

[Grice, 1975] H. Paul Grice. Logic and conversation. In Peter Cole and Jerry L. Morgan, editors, *Syntax and Semantics 3: Speech Acts*, pages 41–58. Academic Press, Inc., New York, 1975.

[Halsey, 1962] William Halsey, editor. *Collier's Encyclopedia*. The Crowell-Collier Publishing Company, 1962.

[Hovy, 1988a] Eduard H. Hovy. *Generating Natural Language Under Pragmatic Constraints*. Lawrence Erlbaum Associates, Inc., Hillsdale, New York, 1988.

[Hovy, 1988b] Eduard H. Hovy. Two types of planning in language generation. In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, pages 179–186, Buffalo, New York, 1988.

[Hovy, 1990] Eduard H. Hovy. Pragmatics and natural language generation. *Artificial Intelligence*, 43(2):153–197, 1990.

[Jameson, 1988] Anthony Jameson. But what will the listener think? belief ascription and image maintenance in dialog. In Alfred Kobsa and Wolfgang Wahlster, editors, *User Models in Dialog Systems*. Springer Verlag, Berlin-New York, 1988.

[Johnson, 1970] Ronald E. Johnson. Recall of prose as a function of the structural importance of the linguistic units. *Journal of Verbal Learning and Verbal Behavior*, 9(1):12–20, 1970.

[Joshi, 1987] Aravind K. Joshi. Generation - a new printier of natural language processing? In *TINLAP-3*, pages 202–205, 1987.

[Kass and Finin, 1988] Robert Kass and Tim Finin. Modeling the user in natural language systems. *Computational Linguistics*, 14(3):5–22, 1988.

[Kass and Finin, 1989] Robert Kass and Tim Finin. The role of user models in cooperative interactive systems. *International Journal of Intelligence Systems*, 4(1):81–112, 1989.

[Kass, 1987] Robert Kass. Implicit acquisition of user models in cooperative advisory systems. Technical Report MS-CIS-87-05, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, Pennsylvania, 1987.

[Konolige, 1981] K. Konolige. A first-order formalisation of knowledge and action for a multi-agent planning system. In J.E. Hayes, D. Michie, and Y.H. Pao, editors, *Machine Intelligence 10*. Chichester: Ellis Horwood, 1981.

[Koubarakis *et al.*, 1989] Manolis Koubarakis, John Mylopoulos, Martin Stanley, and Alex Borgida. Telos: features and formalization. Knowledge representation and reasoning 4, University of Toronto, February 1989.

[Litman, 1986] Diane J. Litman. Linguistic coherence: a plan-based alternative. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 215–223, New York, 1986.

[MacLeod, 1988] Colin M. MacLeod. Forgotten but not gone: savings for pictures and words in long-term memory. *Journal of Experimental Psychology: Learning, Memory and Cognition*, 14(2):195–212, 1988.

[Mann and Matthiessen, 1983] William C. Mann and Christian Matthiessen. Nigel: A systemic grammar for text generation. Technical Report RR-83-105, Information Sciences Institute, Marina del Rey, California, 1983.

[Mann and Thompson, 1987] William C. Mann and Sandra A. Thompson. Thetorical structure theory: A theory of text organization. In Livia Polanyi, editor, *The Structure of Discourse*. Ablex Publishing Corporation, Norwood, New Jersey, 1987.

[Mann, 1983a] William C. Mann. An overview of the nigel text generation grammar. In *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics*, pages 79–84, Cambridge, Massachusetts, June 1983.

[Mann, 1983b] William C. Mann. An overview of the penman text generation system. In *Proceedings of the National Conference on Artificial Intelligence*, pages 261–265, Washington D.C., August 1983.

[Mann, 1987] William C. Mann. What is special about natural language generation research? In *TINLAP-3*, pages 227–231, 1987.

[Maybury, 1990] Mark T. Maybury. The four forms of explanation presentation: description, narration, exposition, and argument. In *Proceedings of the AAAI-90 Workshop on Explanation*, Boston, Massachusetts, 1990.

[McCoy, 1985] Kathleen F. McCoy. *Correcting object-related misconceptions*. PhD thesis, University of Pennsylvania, Philadelphia, Pennsylvania, 1985.

[McCoy, 1986] Kathleen F. McCoy. The romper system: responding to object-related misconceptions using perspective. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 97–105, New York, 1986.

[McCoy, 1988] Kathleen F. McCoy. Reasoning on a highlighted user model to respond to misconceptions. *Computational Linguistics*, 14(3):52–63, 1988.

[McDonald, 1986] David D. McDonald. Description directed control: its implications for natural language generation. In Barbara J. Grosz, Karin Sparck Jones, and Bonnie L. Webber, editors, *Readings in Natural Language Processing*, pages 519–537. Morgan Kaufmann Publishers, Inc., Los Altos, California., 1986.

[McDonald, 1988] David D. McDonald, editor. *Natural Language Generation Systems*. Springer-Verlag, New York, 1988.

[McKeown *et al.*, 1985] Kathleen R. McKeown, Myron Wish, and Kevin Matthews. Tailoring explanations for the user. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 794–798, Los Angeles, California, 1985.

[McKeown, 1982] Kathleen R. McKeown. *Generating Natural Language Responses to Questions About Database Structure*. PhD thesis, University of Pennsylvania, Philadelphia, 1982.

[McKeown, 1985] Kathleen R. McKeown. *Text Generation : Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press, New York, 1985.

[McKeown, 1986] Kathleen R. McKeown. Discourse strategies for generating natural-language text. In Barbara J. Grosz, Karin Sparck Jones, and Bonnie L. Webber, editors, *Readings in Natural Language Processing*, pages 479–499. Morgan Kaufmann Publishers, Inc., Los Altos, California., 1986.

[McKeown, 1990] Kathleen R. McKeown. User modeling and user interfaces. In *Proceedings of the 8th National Conference on Artificial Intelligence*, pages 1138–1139, Boston, Massachusetts, July 1990.

[Moore and Swartout, 1989] Johanna D. Moore and William R. Swartout. A reactive approach to explanation. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1504–1510, Detroit, Michigan, August 1989.

[Moore, 1989] Johanna D. Moore. *A reactive approach to explanation in expert and advice-giving systems*. PhD thesis, University of California, Los Angeles, 1989.

[Neches *et al.*, 1985] Robert Neches, William R. Swartout, and Johanna D. Moore. Enhanced maintenance and explanation of expert systems through explicit models of their development. *IEEE Transactions on Software Engineering*, SE-11(11), 1985.

[Paris, 1987] Cecile L. Paris. *The use of explicit user models in text generation: Tailoring to a user's level of expertise*. PhD thesis, Columbia University, 1987.

[Paris, 1988] Cecile L. Paris. Tailoring object descriptions to a user's level of expertise. *Computational Linguistics*, 14(3):64–78, 1988.

[Perrault *et al.*, 1978] C. Raymond Perrault, James F. Allen, and Philip R. Cohen. Speech acts as a basis for understanding dialogue coherence. In *TINLAP-2*, pages 125–132, University of Illinois at Urbana-Champaign, Illinois, July 1978.

[Prince, 1981] Ellen F. Prince. Toward a taxonomy of given – new information. In Peter Cole, editor, *Radical Pragmatics*, pages 223–255. Academic Press Inc., New York, 1981.

[Rich, 1979] Elaine A. Rich. User modeling via stereotypes. *Cognitive Science*, 3(4):329–354, 1979.

[Sarantinos and Johnson, 1990] Efstratios Sarantinos and Peter Johnson. Explanation dialogues: a computational model of interpreting questions and generating tailored explanations. In *Proceedings of the 5th Workshop on Explanations*, Manchester University, April 1990.

[Sarantinos and Johnson, forthcoming] Efstratios Sarantinos and Peter Johnson. Explanation dialogues: a theory of how experts provide explanations to novices and partial experts. *Artificial Intelligence*, forthcoming.

[Sarner and Carberry, 1988] Margaret H. Sarner and Sandra Carberry. A new strategy for providing definitions in task-oriented dialogues. In *Proceedings of the International Conference on Computational Linguistics*, pages 567–572, Budapest, Hungary, 1988.

[Sarner and Carberry, 1990] Margaret H. Sarner and Sandra Carberry. Tailoring explanations using a multifaceted user model. In *Proceedings of the Second International Workshop on User Models*, Honolulu, Hawaii, March 1990.

[Seamon, 1980] John G. Seamon. *Memory and Cognition*. Oxford University Press, New York, 1980.

[Searle, 1975] John R. Searle. Indirect speech acts. In Peter Coles and Jerry L. Morgan, editors, *Syntax and Semantics 3: Speech Acts*, pages 59–82. Academic Press, Inc., New York, 1975.

[Tversky, 1977] A. Tversky. Features of similarity. *Psychological Review*, 84:327–352, 1977.

[van Beek, 1987] Peter G. van Beek. A model for generating better explanations. In *Proceedings of the 25th Annual Meeting of the Association for Computational Linguistics*, pages 215–220, Stanford, CA, 1987.

[Wahlster and Kobsa, 1989] Wolfgang Wahlster and Alfred Kobsa. User models in dialog systems. In Alfred Kobsa and Wolfgang Wahlster, editors, *User Models in Dialog Systems*, chapter 1, pages 4–34. Springer-Verlag, Berlin, 1989.

[Wilensky *et al.*, 1987] Robert Wilensky, James Mayfield, Anthony Albert, David Chin, Charles Cox, Marc Luria, James Martin, and Dekai Wu. Uc - a progress report. Technical Report 303, Computer Science Division, University of California at Berkeley, 1987.

[Wolz, 1990a] Ursula Wolz. The impact of user modeling on text generation in task-centered settings. In *Proceedings of the Second International Workshop on User Models*, Honolulu, Hawaii, March 1990.

[Wolz, 1990b] Ursula Wolz. An object oriented approach to content planning for text generation. In *Proceedings of the Fifth International Workshop on Natural Language Generation*, pages 95–104, Dawson, Pennsylvania, June 1990.

[Woods, 1970] W.A. Woods. Transition network grammars for natural language analysis. *Communication of the ACM*, 13(10):591–606, 1970.