

**VoiceDBC:
A semi-automatic tool
for writing speech
applications**

**Stephen Derek Choularton
B.Sc., F.I.D.**

**A thesis submitted in partial fulfilment of the requirements for the
Honours Degree in Computer Science in the Division of Information
and Communication Sciences, Macquarie University**

Abstract

One day, keyboards will only exist in museums. The long period of development to computers enhanced with (functional) language understanding is still only just beginning but recent advances have brought us to a point where the question of how to make it easy to build good speech applications has become come to the fore. The advanced state of text-to-speech modules contrasts starkly with the limits of speech recognition, which is still in its infancy. Particular care is required to forget the industry hype, and instead produce applications that can deliver what they are really able to promise.

This thesis provides a review of the existing work related to task-oriented dialogues, good dialogue practice, dialogue phenomena, interfacing with databases and the use of tools in the design of speech applications. It explores the role of good dialogue practice in allowing us to achieve the central objective of producing speech applications that are effective, easy to use and not prone to error. It looks at a wide range of characteristic dialogue phenomena and examines how they can be exploited to support that central objective. It reviews a number of tools that are already available to help write speech applications.

The thesis develops the concept of dialogue design patterns and dialogue-task distance to a point where a new paradigm can be adopted in the design of speech applications. VoiceDBC, a semi-automatic tool for writing speech applications, is the practical outcome of this work. It allows users to build speech applications quickly and effectively and with due regard to the limits of speech recognition. Its design, and implementation are covered in full together with pointers to further work both on the tool, and in the area generally. VoiceDBC can be used to produce simple speech applications in minutes, and handle all database connectivity issues. Tools such as VoiceDBC have their part to play in ensuring that the promise of this field is realized.

Acknowledgements

In everything I do, I have to thank my wife, Elizabeth, for putting up with me and (during the last few years) putting the bread on our table. Without her efforts there would be no letters after my name.

I would like to thank Robert Dale, my supervisor, for keeping me on the straight and narrow and teaching me the little I know about proper academic behavior. The idea of using design patterns in this work was sparked by a throwaway comment of his. Without it, little sense could be made of a semi-automatic tool like VoiceDBC.

Many other people have contributed to the fact that a wrinkly like me (I am 53 years old), who never even matriculated, ended up doing honours. I would like to thank Macquarie University for having a Jubilee Scheme so that wrinklies can access tertiary education without having to matriculate. I would like to thank Carolyn Kennett, without whom I know I would not have passed first year maths. I would like to thank Georgio Martignoni who has provided me with the most practical of help in solving both hardware and software problems and, perhaps more importantly, has given me his friendship. I would like to thank Josef Meyer who always seemed to be there to pull me out of the holes I dig myself into when coding in Perl.

I would like to thank Motorola, Inc. for awarding me the Motorola Language Technology Program Honours Scholarship 2002. This practical support during my honours' year was most welcome.

For everyone else that I have failed to mention by name, thank you quite sincerely for helping me get to this point.

Contents

Page

Chapter 1 - VoiceDBC	i
1.1 Introduction	1
1.2 Task-oriented Dialogues.....	2
1.3 VoiceXML Gateways	3
1.4 Aims, Relevance and Significance of the Project.....	4
1.4.1 Aims	4
1.4.2 The Speech Application Task	5
1.4.3 Relevance	5
1.5 Thesis Overview	6
Chapter 2 – Related Work	7
2.1 Introduction.....	7
2.2 The Application Space.....	7
2.3 The Literature	8
2.4 Good Dialogue Practice	8
2.4.1 Introduction.....	8
2.4.2 The Gricean Maxims	10
2.4.3 Complexity.....	10
2.4.4 Prompts And Re-prompts	10
2.4.5 Vocabulary	11
2.4.6 Lists and Summaries.....	11
2.4.7 Enforcing Good Dialogue Practice.....	12
2.5 Building Speech Applications	12
2.5.1 Introduction.....	12
2.5.2 Analysis.....	12
2.5.2.1 Research the Application Domain.....	13
2.5.2.2 Research the Data Source	13
2.5.3 Design.....	13
2.5.3.1 Dialogue Flow Charts	13
2.5.3.2 Other Design Aspects.....	14
2.5.4 Monitor and improve	14
2.6 Dialogue Phenomena.....	15
2.6.1 Mimicking style	15
2.6.2 Talking to Your Audience	15
2.6.3 Linearity.....	15
2.6.4 Time Out.....	15
2.6.5 Turn Taking	16
2.6.6 Ellipsis	16
2.6.7 Indirect Speech Acts.....	16
2.6.8 Stochastic Variation of Output.....	16
2.6.9 Adjacency Pairs	16
2.6.10 Insertion Sequences	17
2.6.11 Anaphoric References and Referring Expressions Generally	17
2.6.12 Temporal Reference Resolution	17
2.6.13 Disfluencies.....	18
2.6.14 Correction of Errors.....	18
2.6.15 Confirmations.....	18
2.6.16 Help	19

2.7	Database Interfacing Issues	19
2.7.1	Introduction.....	19
2.7.2	Response Generation	19
2.7.3	Database Updates	20
2.7.4	Meta-Knowledge Questions.....	20
2.8	The Use Of Tools in the Design of Speech Applications.....	20
2.9	Conclusion.....	23
Chapter 3 – Dialogue Design Patterns.....		25
3.1	Introduction	25
3.2	Dialogue Design Patterns.....	25
3.2.1	Design Patterns	25
3.2.2	Finding Patterns	26
3.2.3	Using Dialogue Design Patterns in Practice.....	26
3.3	Plane Timetable Application.....	28
3.3.1	Introduction.....	28
3.3.2	The Basic Dialogue Design Pattern.....	28
3.3.3	Using this dialogue design pattern elsewhere	29
3.4	Conclusions	29
Chapter 4 – The Implementation		31
4.1	Introduction.....	31
4.2	The General Architecture.....	32
4.3	Implementing the Design Patterns	33
4.4	Incorporating Good Dialogue Practice.....	33
4.4.1	Introduction.....	33
4.4.2	Forms in VoiceXML	34
4.4.3	Form Specification.....	34
4.4.4	Fields	34
4.4.5	Grammars	36
4.4.6	Blocks	37
4.4.7	Application Wide Rules.....	37
4.5	Coping with the Unrecognisable	38
4.5.1	Introduction	38
4.5.2	Proper Names	39
4.5.4	Emails	39
4.6	Incorporating the Characteristic Dialogue Phenomena	39
4.6.1	Introduction.....	39
4.6.2	Talking to Your Audience	40
4.6.3	Linearity.....	40
4.6.4	Time Out.....	41
4.6.5	Turn Taking	41
4.6.6	Ellipses.....	41
4.6.7	Indirect Speech Acts.....	41
4.6.8	Adjacency Pairs	41
4.6.9	Insertion sequences	42
4.6.10	Anaphoric references	42
4.6.11	Temporal Reference Resolution.....	42
4.6.12	Disfluencies.....	42
4.6.13	Correction of Errors.....	42

4.6.14	Confirmations.....	43
4.6.15	Help	43
4.7	Database Interfacing Issues	43
4.7.1	Introduction.....	43
4.7.2	The Terse Approach.....	44
4.7.3	Lexicons	44
4.7.4	Grammars	44
4.8	Conclusions	44
Chapter 5 - VoiceDBC in Action.....		45
5.1	Introduction.....	45
5.2	The Restaurant Take-away Menu Application	45
5.2.1	The Nature of the Restaurant Take-away Menu Application.....	45
5.2.2	Using VoiceDBC to Write the Dialogue.....	48
5.3	The Catalogue Sales Application.....	53
5.4	Conclusions	53
Chapter 6 – Conclusions		54
6.1	Outcomes	54
6.2	Dialogue Design Patterns.....	54
6.3	A Visual Front End for VoiceDBC.....	55
6.4	Expanding the Library of Templates.....	55
6.5	Implementing Database Connections to Other DBMS's.....	55
6.6	Completing the Work on Incorporating Good Dialogue Practice and Dialogue Phenomena	55
6.7	Conclusions	56
References		57
Appendix I CODIAL: the guidelines for cooperative dialogue.....		61
Appendix II – The VoiceDBC Tutorial		63

List of Figures

Figure 1	<i>VoiceXML document</i>	3
Figure 2	<i>Typical architecture for a VoiceXML application</i>	4
Figure 3	<i>Scope of the work involved in writing a voice application that accesses a database</i>	5
Figure 4	<i>Top voice applications today</i>	7
Figure 5	<i>A screen print of the main Audium interface</i>	21
Figure 6	<i>A screen print of the main CSLU interface</i>	22
Figure 7	<i>A screen print of the main Suede interface</i>	22
Figure 8	<i>A screen print of the main VoiceGenie IDE interface</i>	23
Figure 9	<i>A dialogue taxonomy</i>	27
Figure 10	<i>The basic design pattern for a plane timetable application</i>	28
Figure 11	<i>VoiceDBC, the opening screen</i>	31
Figure 12	<i>VoiceDBC revealing its multi-document text editor</i>	32
Figure 13	<i>Approaches for separating interfaces from code</i>	33
Figure 14	<i>A Mixed Initiative Form</i>	35
Figure 15	<i>An ABNF Grammar</i>	36
Figure 16	<i>The Mixed Initiative Form</i>	45
Figure 17	<i>The Offer Form</i>	46
Figure 18	<i>The Confirmation Form</i>	47
Figure 19	<i>The Review the Data Generally Form from the New Project Wizard</i>	49
Figure 20	<i>The Review 'courses' Form from the New Project Wizard</i>	50
Figure 21	<i>The Lexicalise 'courses' Form from the New Project Wizard</i>	50
Figure 22	<i>The 'Transform' 'courses' Form from the New Project Wizard</i>	51
Figure 23	<i>The Review 'price' Form from the New Project Wizard</i>	51
Figure 24	<i>The Choose the Data To Be Offered Form from the New Project Wizard</i>	52
Figure 25	<i>The final documents being displayed by VoiceDBC</i>	53

The CD Accompanying this Thesis

In keeping with the modern trend a CD accompanies this thesis (fixed to the back cover). It contains a text file holding the code for VoiceDBC.pl (the main part of the program for this application) together with a working, compiled version of VoiceDBC. As is standard practice the CD contains a readme.txt which is reproduced below:

Readme.txt:

Welcome to VoiceDBC. This version requires Windows.

To load VoiceDBC you should drag the entire directory VoiceDBC_V1 from the CD and drop it onto your C drive.

Once completed all you have to do to run VoiceDBC is open C:\VoiceDBC_V1 with Windows Explorer and double click on voicedbc.exe or type c:\VoiceDBC_V1\voicedbc at the command line.

VoiceDBC was developed on a computer with screen settings of 1024 x 864 and views better on such larger settings.

This version of VoiceDBC is set up to use Microsoft's Access DMBS so you have to have Microsoft Access on your computer for it to work fully. If you do not you will still be able to see the applications it has already written which are shipped along with it.

In order to make it easier to review this version without any training the New Project Wizard has many fields set to default to the values that are used in the Tutorial.

The uncompiled code for VoiceDBC can be found in the file VoiceDBC.pl in the root directory of this CD.

Any problems please contact me, Stephen Choularton, at stephenc@ics.mq.edu.au or stephen@bymouth.com

Chapter 1 - VoiceDBC

1.1 Introduction

This project is concerned with the question of how to make it easy to build good speech applications. Over the last few years, major advances have been made in the field of speech technology. Text-to-speech¹ is now highly developed. Speech recognition, if used carefully, can allow a computer to handle a wide range of utterances. Companies, such as Microsoft and Apple, are providing Speech Engine modules that can be integrated with their operating systems. Programming languages such as C++ and Visual Basic now come with Speech APIs. Nuance, Speechworks, Carnegie-Mellon University and others provide speech engine components and even full VoiceXML Gateways that can be downloaded free for development purposes. Some are even open-source.

High level dialogue modeling languages such as VoiceXML (an international standard), and SALT (an industry standard) have brought the field to a stage similar to that of the Internet just after HTML and Perl CGI programming became common. Suddenly it was possible for anyone (with a modicum of skill), to write the HTML to produce a website and the Perl code that enabled it to talk to a database. Such applications make up a large slice of the World Wide Web but making the task easier did nothing for the quality of websites produced. Indeed many today still leave much to be desired with poor quality interfaces that lack any intuitive feel (including difficulties in navigation, different responses to the same actions and the like), give poor feedback, and use poor English.² Very similar problems are being faced with speech applications.

Writing speech applications requires many unassociated skills in the fields of language technology, internet technology, network technology and database technology. Some of these skills are based upon a notion of *good dialogue practice*, which is still in its infancy. A good website can be characterised as adopting *good website practice* and a working definition for good dialogue practice is: *Those strategies and tactics which, when adopted in the designing of a dialogue, produce speech applications that are effective, easy to use and not prone to error.* In consequence, an important area of research for this project concerned the literature relating to good dialogue practice. In turn, much of this relies on exploiting the phenomena that characterise dialogues, such as turn taking, so a second important area was the literature concerning dialogue phenomena.

Recent advances in support for the construction of speech applications have resulted in a number of tools that allow a developer to build an application by visually arranging components on a canvas, or by selecting alternatives from menus. However, none of these applications provide support for building applications that adhere to good dialogue practice. To do so applications must either deliver compliance with good dialogue practice as a designed in feature or provide support similar to that provided by grammar-checkers or spell-checkers in their respective fields. None of the applications reviewed, for example, require developers to limit the length of prompts, produce correct referring expressions, choose appropriate vocabularies, or give proper help. Nor do they provide supporting modules relevant to common dialogue phenomena such as resolving anaphoric or deictic temporal references, or other common anaphora.

Following research into many existing applications and the literature relating to good dialogue practice, dialogue phenomena, development of speech applications and natural

¹ Where necessary these domain relevant expressions will be explained further in this chapter.

² I am unable to comment on those written in other languages.

language interfaces with databases, I produced a tool; *VoiceDBC*³ aims to incorporate good dialogue practice, contains modules to handle a chosen range of dialogue phenomena and eases the problems of interfacing with databases.

In this chapter, we firstly look at task-oriented dialogues, the particular type of speech applications relevant to us. In order to put the work in context, I chose VoiceXML Gateways as the platforms for which VoiceDBC would write speech applications and, so next, we look at VoiceXML Gateways. Then we look at the aims, relevance and significance of this project, and finally give a thesis overview.

1.2 Task-oriented Dialogues

Speech applications are diverse. They can range from dictations systems (like the well-established Dragon dictation system) to voice dialing modules on mobile phones, from car navigation systems to horse race betting systems. The particular area of interest for this project lies in speech applications that handle *task-oriented* dialogues. The expression task-oriented is becoming increasingly used in the literature to define the space that many speech applications written today occupy. Balentine and Morgan [1999] define a task-oriented dialogue as:

“A dialogue concerned with a specific object, aiming at a specific goal (such as resolving a problem or obtaining specific information). For example, dialogues concerned with flight information, email access or transferring funds are task-oriented.”

One also finds the expressions ‘task-focussed nature’ in Beasley *et al.* [2002], ‘task-oriented’ in DISC [1999,2000]) and again in Mittendorf [2001] and elsewhere in the literature. The following sample dialogue (a timetable query) is typical of what lies fairly and squarely within our domain.

Computer: Good Morning, I can help you plan a flight. Where do you want to go?
Person: Sydney.
Computer: Where are you flying from?
Person: Darwin.
Computer: What day do you want to depart on?
Person: Thursday.
Computer: Please hold while I check the database.
Computer: OK, there is a flight that departs Sydney at 9:50 am and arrives Darwin 6:50 PM. Do you want a later flight?
Person: Yes.
Computer: OK, there is a flight that departs Sydney at 12:50 am and arrives Darwin 9:50 PM. Do you want a later flight?
Person: No.
Computer: Can I help you with any other enquiry?
Person: No.
Computer: I hope we can help you in the future. Goodbye.

I have concentrated exclusively on task-oriented dialogues as, given the current state of development of this language technology, normally it is only these simple dialogues that can be successfully handled by a computer. It is extremely difficult to write a speech application where the developer does not know that a relatively simple pre-defined outcome is required. The time will come when more profound levels of language

³The DBC stands for *database connectivity*.

```

<?xml version = "1.0"?>
<vxml version = "2.0">
  <script>
    function greeting() {
      /* create a new date object */
      var today = new Date();
      /* figure out if is morning or not */
      var hour = today.getHours();
      var response = "Good Morning";
      if (hour > 11) {response = "Good Afternoon";}
      if (hour > 17) {response = "Good Evening";}
      response = response + ".";
      return response;
    }
  </script>

  <form>
    <field name = "answer">
      <prompt>
        <value expr = "greeting()"/>. Would you like
          me to help you plan a flight?.
      </prompt>
      <grammar>
        yes | no
      </grammar>
    </field>
  </form>
</vxml>

```

Figure 1 – A VoiceXML document

understanding change this but for the moment I have made no attempts to move out of the area.

1.3 VoiceXML Gateways

This project leads to the implementation of a tool that writes speech applications. Those applications have to be run somewhere and I chose VoiceXML Gateways as the platforms for which VoiceDBC would write these speech applications.

While there are some 250 million computers connected to the Internet, there are some 1.3 billion phones that are now capable of accessing the Internet through any voice gateway. However, it is important not to confuse this use of the Internet with the World Wide Web. It is possible to write applications that scrape text off web pages and then render it to speech and to speech-enable web-surfing, but voice gateways are more interested in crossing the bridge between telephony and computing, and in using the Internet's ability to allow applications to use remote resources. Normally, this will be by way of accessing a remote database or by obtaining a dynamically created VoiceXML document.

Voice gateways, (also known as voice portals) are bundles of software sitting on a computer that can accept incoming phone calls. The computer can apply speech recognition to incoming sound and, if it is a VoiceXML Gateway, pass the resultant text to a VoiceXML Browser for interpretation. Resources can be commanded from the Internet and the browser can produce text outputs that can then be rendered by a text-to-speech module and sent back down the phone line. Other voice gateways might use SALT, C++ or any one of a number of other programming languages to handle the dialogue. In theory, voice gateways could allow access to the Internet to increase many fold with access being as easy for those who are computer literate as for those who are

not. In addition, the facility can be of great help to those who are technologically or physically deprived. The implications for the sight-impaired are obvious.

A significant development in the field is the highly level dialogue description language called VoiceXML 2.0. It complies with XML's strict guidelines and fills a similar role to HTML on the 'visual' web. ECMAScript (a formal specification of the 'visual' web's JavaScript) is used as the scripting language. Data items inside VoiceXML are ECMAScript variables and can be manipulated with all the expected programming control structures and assignment operations. An example of these interactions between VoiceXML and ECMAScript is shown at *Figure 1*. This code results in the system providing a greeting whose content depends on the time of day. If the person responding says yes, the variable *answer* takes on the value yes (as allowed by the grammar *-yes | no*).

A typical architecture for an application in this field might be as shown in *Figure 2*. Anyone with the correct phone number (and possibly an access code) can dial in and start talking to the voice portal. If the dialogue is self-contained, no access to the Internet occurs, but, if some information or dynamic response is required, the voice browser will use the submit action (based on the http protocol) in much the same way as a visual browser would. The receiving server will fire up some code held inside its cgi-bin which will either produce a dynamic response (send back a new VoiceXML document for the browser to interpret), or access a database (on the server or more remotely over the Internet), and then produce a dynamic response

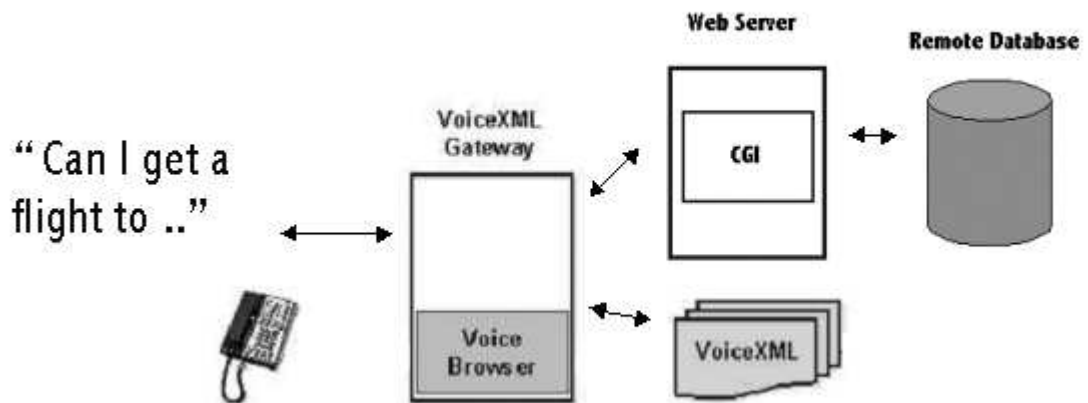


Figure 2 – A typical architecture for a VoiceXML application

1.4 Aims, Relevance and Significance of the Project

1.4.1 Aims

This project is concerned with the question of how to make it easy to build good speech applications. To do this, it:

- studies other applications for writing speech applications;
- collects and collates material on:
 - good dialogue practice;

- dialogue phenomena;
- development of speech applications; and,
- natural language interfaces with databases.

Using this knowledge it then specifies, designs and implements an integrated development environment – called VoiceDBC.

1.4.2 The Speech Application Task

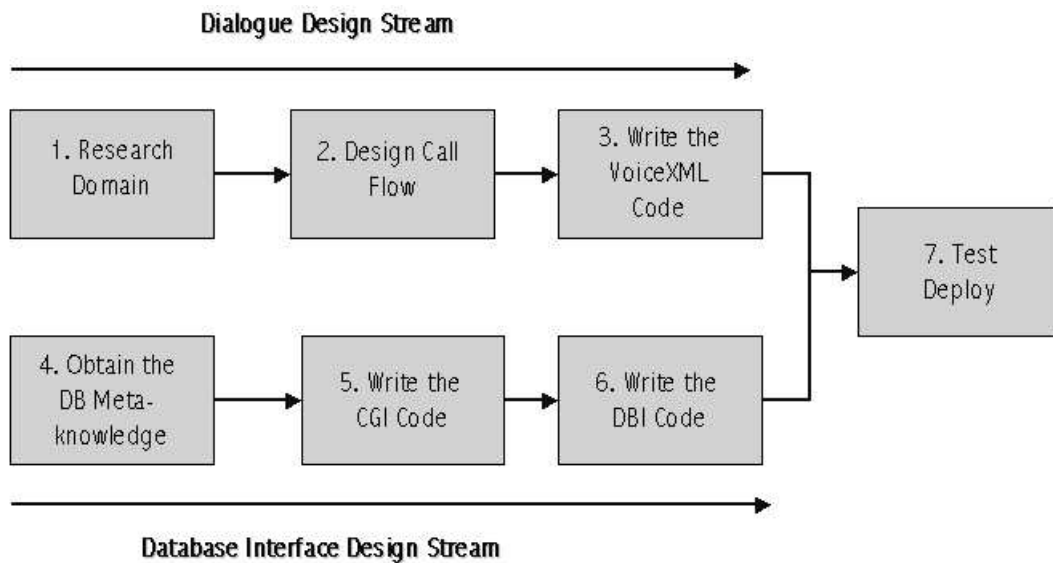


Figure 3 – Scope of the work involved in writing a voice application that accesses a database

The scope of work involved in writing voice applications that access databases is summarised in Figure 3.

- The developer must **research the domain** (*Step 1*). This involves looking at what is done naturally in the domain and sometimes running simulations of dialogues.
- At the same time, the developer must obtain **the database meta-knowledge** (*Step 4*) – the table names, the field names, the nature and scope of the data contained therein.
- Next a **design call flow** (*Step 2*) must be created, planning the course the dialogue should take, accounting for any diversions from the predicted dialogue, and bearing in mind the underlying form of the data.
- Next **the VoiceXML code** (*Step 3*) must be written.
- Next **the CGI code** (*Step 5*) must be written.
- Next **the DBI code** (*Step 6*) must be written.
- Finally, the whole must be **tested and deployed** (*Step 7*).

VoiceDBC aims to automate, as far as possible, all but *Step 1* – **research the domain** - and *Step 7* – **test and deploy**.

1.4.3 Relevance

In the course of the next few years, voice portals will roll out across the world. Many (perhaps most) of the applications they run will involve a database as a back end. The

easy production of voice applications capable of conforming to good dialogue practice and operating in that environment is highly relevant.

1.4.4 Significance

It is very timely to review and collate the diverse work done by others on good dialogue practice. Speech applications are at a point, where the inability of speech recognition to deliver what the industry hype promises, requires a very considered response. In addition, there has been very little serious input to the question of handling many of the phenomena that characterise task-oriented dialogues. Of course, any tool that actually delivers the promise of making it easy to produce good speech applications, will be of immediate and practical use.

1.5 Thesis Overview

In the balance of this thesis, we will look at the work done on this project. *Chapter 2* reviews the related work. This consists of a large number of existing applications which fall within our application space, a very considerable literature at both an academic and practitioner level, and finally, a number of existing software tools that assist developers to write speech applications. *Chapter 3* looks at dialogue design patterns. Perhaps one of the most interesting concepts to emerge from this project, dialogue design patterns allowed me to adopt a new paradigm, that avoids the user being required to undertake a design call flow, (*Step 2 of Figure 3*) and that cuts short the traditional application development life-cycle. *Chapter 4* goes into detail about the implementation. It explains how dialogue design patterns are used in VoiceDBC, how good dialogue practice was incorporated into VoiceDBC, and which of the characteristic dialogue phenomena VoiceDBC supports. It explains the general approach to coping with writing the VoiceXML code, obtaining the database meta-knowledge, writing the CGI code and writing the DBI code (*Steps 3 to 6 of Figure 3*). *Chapter 5* looks at the actual use of VoiceDBC to write an application (*The Restaurant Take-away Menu Application*). It looks at the nature of this application, and then takes us through writing it using VoiceDBC, concluding the work on *Steps 3 to 6* (of *Figure 3*). Finally, *Chapter 6*, the *Conclusion*, reviews the outcomes of this project and gives pointers to future work.

Chapter 2 – Related Work

2.1 Introduction

This chapter discusses the related work leading up to this project. Firstly, *Section 2.2* explores the application space, looking at the considerable number of applications which have been produced in the academic and commercial domains. Next, it reviews the relevant literature relating to good dialogue practice (*Section 2.4*), building speech applications (*Section 2.5*), dialogue phenomena (*Section 2.6*) and interfacing with databases (*Section 2.7*). Then it reviews some of the existing tools that can be of use to developers of speech applications (*Section 2.8*). Finally, in the conclusions, it explains how all this work leads to VoiceDBC.

2.2 The Application Space

As pointed out earlier, (at *Section 1.2* above) speech applications are very diverse. The particular area of interest to this project lies in speech applications that handle task-oriented dialogues. Hood [2002] provides us with a list of existing spoken dialogue systems, many of which fall within this domain. It covers some 62 different projects, many of which have been developed experimentally by university departments. About half of these fell clearly into our application space. It is clear that a very considerable amount of work has been done; some ten (nearly a third) of the applications were to do with travel, covering from flight reservations to train times; the balance were across a wider range with, *inter alia*, weather information, securities trading, restaurant information, email access, traffic information and classified adverts represented.

The commercial field has also been active. *Figure 4* reproduces a table from the 2001 VoiceXML Forum, listing the top commercial applications.

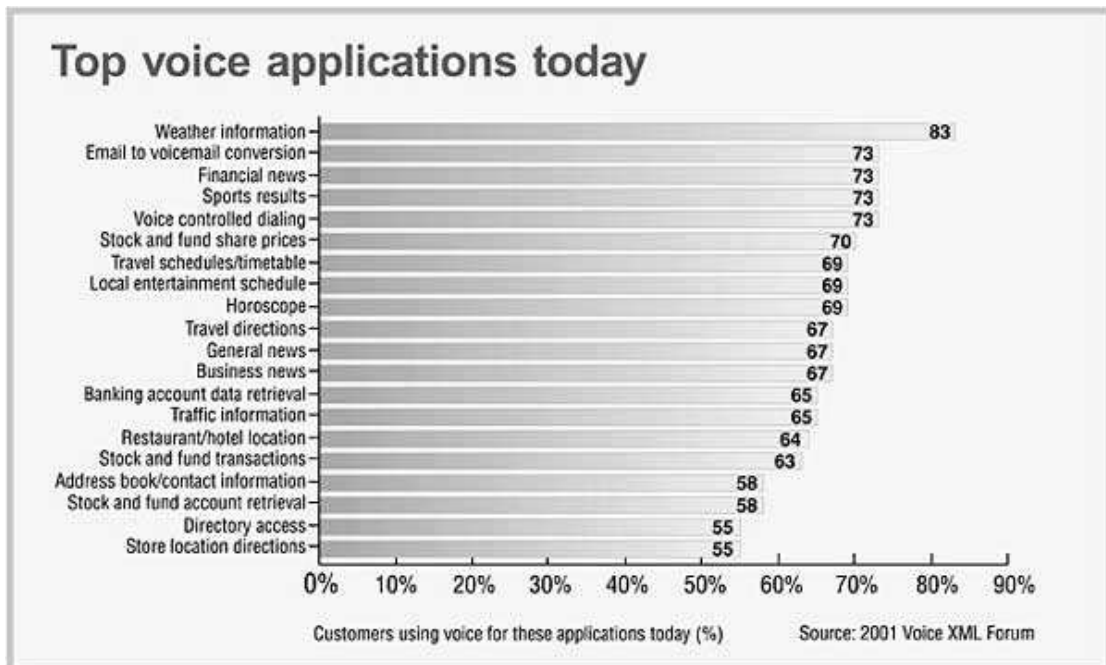


Figure 4 – Top voice applications today

All of the commercial applications (other than voice controlled dialing) fall within the application space of interest to this project. The subject matter of the commercial

projects shows more diversity than their academic counterparts but it would seem that only the surface of what can be handled in the application space has been scratched. This matter is further explored in *Section 3.2* when we will look at a taxonomy for task-oriented speech applications.

2.3 The Literature

The literature falls into two camps, material from practitioners and material from academia. There is a considerable demand from industry and academia for up-to-date information about VoiceXML and voice portals. It is not surprising that a number of practitioners have written book to satisfy that interest (including Abbott [2002], Andersson *et al.* [2001], Beasley *et al.* [2002], Edgar [2001] and Sharma and Kunins [2002]). Without going into great depth, these tend to review the relevant areas, extending from an overview of the architecture of voice portals through, *inter alia*, how to write VoiceXML and EMCAScript code and CGI programming. They give the reader a feel of what is possible and what is happening in the field. They tend to be short on depth about such things as good dialogue practice. However, one practitioners' book that concentrated solely on dialogue, was that by Balentine and Morgan [1999]. It takes the form of an extremely detailed style guide and should be read by anyone wanting to work in this area.

The academic material ranges more widely and touches on a number of areas which, in the end, I classified as good dialogue practice, building applications, characteristic dialogue phenomena and database interfacing issues.

Two sources were websites and, in this field, information published only on the Internet is of growing importance. In my review of the literature, I have used the four classifications above to group common material. It may be that at some points the boundaries between good dialogue practice and characteristic dialogue phenomena blurs. This is because the former often endeavors to exploit the latter to achieve its objective of producing speech applications that are effective, easy to use and not prone to error.

2.4 Good Dialogue Practice

2.4.1 Introduction

What does the literature have to say about good dialogue practice? Clearly, much of it is of the view that good dialogue practice is important. Balentine and Morgan [1999], Beasley [2002], DISC [1999, 2002], Fraser [1997], TRINDI [2001] and others, include guidelines of one form or another. Different authors stress different aspects and some even contradict each other; however, they tend to all agree that one is trying to maximize the beneficial effects of several different factors:

- *Conventions*—The natural desire of humans for consistency is often expressed in the conventions they adopt. It is important that when faced with new applications, users can rely on previous experience. Consistency in style, usage of words, cause and effect (as appropriate) are expected in human-machine interfaces from the arrangement of pedals in a car to graphical user interfaces on a computer screen. For example, most people would expect a large red button on a machine to be an emergency stop button simply because that is the common usage of such a button. This desire for consistency extends to the common use of words from other computer applications. As it happens, 'help' is a word that we might wish to avoid as it is difficult for speech recognisers to recognise. However, the fact that it is so commonly used to raise assistance in computer applications, leads to a seemingly universal recommendation for it be used for the same purpose in speech applications.

One recent approach [Rosenfeld *et al.* 2001] has gone so far as to sacrifice any pretence of natural language processing on the theory that it is better to have complete consistency in the command language used across different applications.

- *Control*—A second factor is the designer’s desire to allow the application to ‘control’ the nature of the user answers by exploiting characteristic dialogue phenomena. These phenomena are dealt with in more detail below but briefly, as people in a conversation tend to copy the style of conversation used by other participants, one practice is to use simple direct prompts to encourage simple, direct replies. *Do you want to travel today* is much more likely to give rise to a “es”No answer than *There is a flight via Hong Kong at 1230 and another via Bangkok at 1130. Do you want to travel today*. Another way of exercising control is to use timing and re-prompting to dampen down corrective interruptions. Given that computers open windows of a limited time duration for the purpose of recognition, miss timing can soon result in the human talking when the window is closed and the computer never having a chance to properly hear the input.
- *Recognisability*—A third factor is the designer’s the desire to encourage the types of response that speech recognisers find most easy to decode. Speech recognisers do not handle words in a manner in which human intuition finds easy. For example, as Abbott [2002] points out, they find it easier to recognise ‘speak louder’ than ‘louder’. This becomes easier to appreciate when one considers the speech recogniser is using probability to decide which word is being said. A phonetic pattern that contains more data will provide more information for matching purposes. Indeed, the most falsely inserted (recognised when not spoken) and falsely detected (spoken but not recognised) word is ‘six’, the shortest spoken digit in the language.
- *Initiative*—A fourth factor that most sources appear to agree upon, is how to place the initiative in the dialogue. The consensus is that the system should be enabled to take general opening responses (at each stage of the dialogue) and then fall back on a system led initiative to obtain any additional information required from the user. For example, when trying to obtain flight information, the system should be able to take what is useful out of an utterance like “I want to fly to Darwin on Thursday.”, and only seek clarification of from where the speaker wants to depart. While this is not really as flexible as the expression *mixed-initiative* implies it is often all that can be achieved.

As Ballentine and Morgan [1999] point out, it is too early to set real standards for good dialogue practice. There is no adequate collection of data to base it upon. However, it is certainly not too early to start on the process of codification. The several factors outlined above will influence the efforts of designers to produce useful approaches to such matters as the handling of large amount of data, recovery from error and help with an application. Many practices will arise unpredictably as designers explore the space that speech applications can occupy. The successful ones will come to be adopted as normal and, as they help a user move from one application to another, they will be expected by the user.

In this section we will look at a number of areas which emerge from the literature as being important to good dialogue practice. These range from the degree of task complexity that task-oriented dialogues should aim to cope with, through appropriate English style and vocabulary, and onto how we can deal with large data sets. Finally, the section turns to how we might enforce good dialogue practice. First we look at the Gricean Maxims.

2.4.2 The Gricean Maxims

No discussion of good dialogue practice in this field could take place without paying respects to the Gricean Maxims. They are explicitly cited in many of the papers reviewed. Grice [1975] looks at how a number of maxims can be used to interpret what is implied by the utterances that occur in a co-operative conversation. It is assumed that the speaker adheres to these maxims:

- Quantity
 - Make your contribution as informative as is required (for the current purpose of the exchange).
 - Do not make your contribution more informative than is required.
- Quality – Try to make your contribution one that is true.
 - Do not say what you believe to be false.
 - Do not say that for which you lack adequate evidence.
- Relevance – Be Relevant.
- Manner – Be Perspicuous.
 - Avoid obscurity of expression.
 - Avoid ambiguity.
 - Be brief (avoid unnecessary prolixity)
 - Be orderly.

They may (almost) be considered a fifth factor acting upon good dialogue practice.

2.4.3 Complexity

Abbott [2002] makes the point that, in designing speech applications, it is better to concentrate on the easier 80 per cent of requests, handling them simply and cost-effectively, keeping response vocabulary simple and generic, and allowing users to fall back onto keypad response or a human operator when significant problems arise. By making prompts simple, one encourages simple replies. At the current stage of speech recognition, a key feature is to encourage users to speak in a way a computer can understand.

According to Glass [1999], Flammia [1998] contains statistics that concern the human side of human-machine dialogues in the movie domain. The study showed that an average dialogue consisted of 28 turns and user queries were not very long (over 80% of user utterances were fewer than 12 words and half, four or less) confirming that task-oriented dialogues are often quite simple in any event..

2.4.4 Prompts And Re-prompts

Most sources (see for example Ballentine and Morgan [1999] at pages 42 onwards) recommend the use of simple, direct prompts consisting in their earlier parts of any required instructions and ending with the ‘call to action’. When the system has to prompt the user a second or subsequent time, (for want of recognition) it should reduce the length of the prompt probably only using the ‘call to action’. Rolling two questions into one (*e.g.* Do you want to fly one Friday and will you want a car?) should be avoided. They recommend that when prompting for Yes/No answers, one should use the interrogative form (*e.g.* “Is this correct?”). Include the verb and only use the imperative form for error-recovery (*e.g.* “Please answer “Yes” or “No”.) With prompts generally, use the interrogative for constrained numerical data (*e.g.* “How many shares?”) but avoid it for unconstrained data (*e.g.* “When were you born?” is better rendered as “What year you were born? What month? What day?”). Further, one should drop the verb when the

interrogative is implied (e.g. “PIN number?”) They recommend that when one wants a verbatim response, use a transitive verb like “say” (e.g. “Say “Sydney” or “Darwin””) and don’t insert extra words after “Say”. When one wants data, use “state” to avoid the user simply parroting back the direct object of the verb phrase. Don’t mix the two forms in one prompt. Should one use “please” and “thank you”? Well, one is trying to maximize the fact one can set the style of responses by the prompts and if one is after terse responses, perhaps not.

2.4.5 Vocabulary

As Ballentine and Morgan [1999] points out, sometimes there are vocabularies already associated with consumer technologies that will be readily understood and spoken by users. These are not just the obvious ones like “help” being associated with computers. Most users will, for example, know “cancel”, “enter” and “OK” from the Automatic-Teller-Machine. This can, of course, conflict with an equally important requirement to pick an utterance that a current speech recogniser will find easy to recognise. Generally, this requirement means choosing multi-syllable words or phrases and avoiding noisy words (containing unvoiced fricatives, ‘f’ and ‘s’) and low-energy words (containing phonemes such as the nasal ‘m’ and ‘n’). Word pairs that have excessive syllable sharing, like “Ice Cream” and “I Scream”, should also be avoided.

Sharma and Kunis [2002] makes a similar point, that some commonly accepted set of user instructions will become the standard. Such a list of user instructions may come sooner than expected. In order to enable the reuse of a user’s knowledge between different applications and devices, the European Telecommunications Standards Institute has published a standardised, minimum generic set of spoken command vocabulary (across five different languages) [ETSI 2002].

2.4.6 Lists and Summaries

Generally, large quantities of data are often presented in tables. Owing to the linear nature of speech, these tables become lists, and long lists bring their own problems. Human memory is short and one effective method offered by Ballentine and Morgan [1999] to make list interruptible and to allow bi-directional navigation. “Stop”, “back-up”, *etc.* can empower the user in respect to long lists.

Large quantities of data can also be summarised. When it comes to the presentation of data, Walker *et al.* [1998] has some practical guidance from research. They look at how the dialogue agent can solve its major problems: *what* information to communicate to a hearer and *how* and *when* to communicate it. The agent may adopt a number of strategies when reading messages and when summarising messages. The choices are:

<i>Read</i>	- Read-First	<i>Summarise</i>	- Summarise-System
	- Read-Choice-Prompt		- Summarise-Choice
	- Read-Summarise-Only		- Summarise-Both

This is just the sort of problem, which VoiceDBC will often face. Walker’s references (unsighted), explain that decision theoretic planning can be applied to the problem of choosing among strategies by associating a utility with each choice and by positing that agents should adhere to the Maximum Expected Utility Principal ([Kennedy and Raiffa 1976] and [Russell and Norvig 1995]). In addition, several reinforcement learning algorithms, such as dynamic programming and Q-learning, specify a way to calculate utility in these matters ([Bellman 1957],[Watkins 1989],[Sutton 1991] and [Barto *et al.* 1995]) which Walker’s paper completes.

The outcomes are perhaps of greater interest to us than the actual mechanics; three experiments were undertaken in which users completed three representative application tasks that required them to access email messages. The results of these experiments were to show that Read-First (that is, where the system commences by reading a message without consent) has the highest user utility. The best summarising strategy (again measured by utility) is the Summarise-System (where the system decides upon the criteria by which to summarise).

Lists also bring with them ‘one anaphora’ management problems that are dealt with further below (at *Section 2.6.11*).

2.4.□ □nforcing Good Dialogue Practice

The only really clear attempt to enforce good dialogue practice found in the literature was a detailed methodology called CODIAL [DISC 1999, 2000]. This involves the recording of Wizard of Oz (Wo□⁴) simulations and their comparison with a checklist of salient points. The results are used to improve the dialogue design and the process is then applied iteratively. This work moves some way towards the ideas developed in this project but does not, for example, look at exploiting the wide range of dialogue phenomena in promoting good dialogue practice.

2.5 Building Speech Applications

2.5.1 Introduction

Beasley [2002], DISC [1999, 2000], Glass [1999], Hulstijn [2000] and Sharam [2002] all suggest that the development of speech applications fit in quite naturally with a commonly used iterative model (iteration within the waterfall model). This model allows some freedom to use feedback between the stages of the development process and expects considerable iteration within each stage. Readers are referred to Somerville [1989] (at page 9) for a fuller description of this development model. The basic stages are:

- Analysis – Normally associated with the production of some form of system requirement specification.
- Design – Normally associated with the production of a formal design document.
- Implementation – Normally associated with the creation of the code.
- Monitor and improve – Normally associated with the application in the field

In this section, we will look at what the literature has to say about building speech applications. First, we will look at the types of analysis that are most suitable for these types of application; then the special flow charts used in design. The literature has nothing very special to say about implementation, but the phase after a speech application has been fielded is unusually important and is considered further.

2.5.2 Analysis

The analysis phase with a speech application concentrates on two areas. First we will look at the work that is necessary on the application domain. For example:

- What sorts of things are said?
- How do dialogues run?

⁴ Wizard of Oz [Wo□] simulations allow the pretence that a human is having a dialogue with a computer. They are based upon the famous encounter with the Wizard of Oz where Dorothy and her entourage believe they are talking to a monstrous wizard, when, in fact, an Omaha sideshowman is pulling the strings of the ‘wizard’ from behind a screen.

Then we will look at the work to be done to enable the speech application to connect to a back-end database.

2.5.2.1 Research the Application Domain

Normally, research into the application domain involves attempts to collect and analyse a corpus of actual dialogues. Much of the literature in this field points to the problem that, with speech applications, one is often not simply trying to automate an existing process. Fraser [1997] and Glass [1999] analyse this aspect in detail. Fraser characterises three types of approach in design of dialogues:

- *Design by intuition* – It is natural for people to think they can construct simple dialogues by intuition. They do it every day, and classic contributors to the field like Chomsky relied heavily on empiricist structuralism (that is, the discovery of the meanings and patterns that already exist in dialogues).
- *Design by observation* – Exploring corpora of spoken language in the process of designing a speech application.
- *Design by simulation* – Often, an application will be doing something that is only similar to an existing activity (for example, phone banking is only similar to dealing with a bank teller). In addition, generally only human-human dialogues exist. A developer actually wishes to model a human-machine dialogue that does not yet exist and for these reasons simulations can actually play a part in the analysis stage and, where resources allow, formal Wo \square simulations can be undertaken to reveal what may occur in these proposed dialogues. Of course, as [DISC, 1999, 2000] points out, most simple dialogues can be handled on an implement-test-and-revise basis – a process falling very short of a Wo \square simulation.

2.5.2.2 Research the Data Source

Few applications can do anything useful without a data source and the literature relating to this is covered in greater depth in the *Section 2.7* below. Clearly, the developer will require to know the nature of the data source. By this I mean not only the size and diversity of the data, but also the field names, the proprietary name of the database, the path to the database, any security requirements, and any special data types used.

2.5.3 Design

2.5.3.1 Dialogue Flow Charts

There seems to be a common agreement that developing speech applications is a highly iterative procedure, not lending itself to formal or waterfall techniques (DISC [1999, 2000], Glass [1999], Hulstij, [2000] and Sharam [2002] amongst others). However, a formal design for the dialogue seems to be accepted as universal. This can take a number of forms from simply writing the dialogue down to Universal-Modeling-Language. However, dialogues are normally viewed as finite state machines and the most popular approach is to use some sort of flow chart that captures the dialogue states and transitions, the data to be obtained and the different dialogue routes to be navigated based upon events during the dialogue. Generally, a modular approach is promoted using ‘sub-dialogues’ to contain and manage the complexities of the overall flow. A sub-dialogue would be called from a main dialogue flow to handle a discrete part of it (*e.g.* in a travel dialogue the hotel booking and car hire sections might each be contained in their own sub-dialogues). Creation of such dialogue flow charts allows the technique of simulation to be used on the specified dialogues with a view to iterative improvement at this early stage. In addition, it allows checklists to be applied (again at an early stage).

These checklists are salient points that can be used by the developer allowing repeated verification of the application (or parts of it) for compliance with good dialogue practice. Many sources include lists of good dialogue practice (see particularly Ballentine & Morgan [1999]), but CODIAL is an actual checklist that is a feature of DISC.

The CODIAL tool consists of a set of rules in support of cooperative dialogue design and a methodology for using them. There are 13 generic guidelines and 11 specific guidelines that are reproduced in *Appendix I*. The first nine come directly from the Gricean Maxims [Grice 1975] of quality, quantity, relevance and manner. The next three generic guidelines expand to address system specific matters: *partner asymmetry*, that is making the user aware that the interlocutor is not a ‘normal’ partner, stopping the generation of all sorts of miscommunications that the computer cannot possibly handle; *background knowledge*, that is making sure the system is sensitive to the background knowledge of the user; and, *meta-communication*, that is, appropriate system behaviour for the purpose of clarification and repair of the dialogue. The last 11 (these are the specific guidelines) give greater detail on the application of previously more generally stated guidelines. The tool comes with instructions on its use as a design guide and its use for diagnostic evaluation. The guidelines were developed against the background of a series of WoM simulations. Essentially, one checks such simulations against the guidelines to ensure compliance, amends the dialogue where errors are occurring and continues this process in an iterative manner.

2.5.3.2 Other Design Aspects

If the developer is using a high-level dialogue specification language like VoiceXML, design decisions will normally be constrained. The basic design of the dialogue will already have been determined by the work done on the dialogue flow charts. However, decisions will be required concerning the design of the back end. The major decision concerns the programming language to be used, with Perl and Java as the main contenders. The developer will have to consider how to access the application data. This has a number of aspects:

- Will the data be available over the Internet, on the same hard drive, or through a Local Area Network?
- Will ODBC, JDBC or some other middle-ware be used for the connection?
- Will Structured Query Language, or some other language, be used to interact with the database?

Finally, the developer will have to provide facilities to convert any data encryptions into natural language (‘bri’ in the database has to end up as ‘Brisbane’ in the speech output).

It is possible to consider the incorporation of other components into VoiceXML applications. Mittendorf *et al.* [2001] concern themselves entirely with the possibilities of incorporating intelligent component technologies into VoiceXML based systems. They review some items where natural language understanding might be desirable. However, they indicate that integration of components is not seen as easy and this may have stopped developers from trying to incorporate these cutting edge technologies into the design of their VoiceXML applications.

2.5.4 Monitor and improve

Once fielded, a speech application will benefit from monitoring and deployment. Ballentine and Morgan [1999], Mankoff [1999], and others deal with the post deployment aspect of voice applications and its importance cannot be stressed enough. The literature

accepts continued monitoring and improvement after the fielding of an application to be particularly important for speech application. So much so that 100% recording of actual use is often recommended together with the collection of a battery of statistics. Two of these are: how many users ring-off before the application completes, and the number of turns taken in each dialogue. This flows on from the fact that human-computer dialogues are a new phenomenon and their true nature can only be discerned by monitoring. This can lead onto a pro-active approach to the adjustment of the dialogue to improve user satisfaction.

2.6 Dialogue Phenomena

Save for anaphora and referring expressions, (which are ‘hard core’ subjects for natural language technology) not many papers concentrate exclusively on one dialogue phenomenon. However, much interesting material is available.

2.6.1 Mimicking style

Mankoff [1999] points to the work of Coltan-Ford [1991] (not sighted), which indicates that humans do tend to mimic a computer they are talking to. This is not surprising. We know from our own experience that there is a tendency for the way people talk, that is, their style of speaking and vocabulary, to converge to a common or dominant style within any particular dialogue.

2.6.2 Talking to our Audience

In human conversation, a person adjusts the way they speak to suit the experience and skills of the listener. This is mirrored in the Gricean Maxims [Grice 1975] and many of the sources make the point that speech applications should incorporate this aspect. Abbott [2002] points out that experienced users do not need too many or too long prompts. Equally, inexperienced ones need extra guidance.

DISC [1999, 2000] goes into some detail on this aspect. Maintenance of proper histories of the interactions with the system can help refine responses for different users. However, even categories like *novice* and *expert* obscure the true nature of users by obscuring the world knowledge that people bring to their use of a system. Expected users might actually be domain or system experts who would require a third style of conversation.

2.6.3 Linearity

As Ballentine and Morgan [1999] point out, speech (and dialogue) is sequential and subject to any deficiencies in human memory. Human dialogues allow for exchanges to remind a participant of what has already been said and speech applications must do the same by allowing features like *repeat*.

2.6.4 Time Out

Human dialogues allow participants to suspend activity for a time to attend to other matters. Abbott [2002] and Andersson *et al.* [2001] both agree that speech application should always offer this facility.

One kind of time out is the requirement to note something down during a conversation. DISC [2000, 1999] points out that systems should be able to deliver information at a slower rate when the user wishes to take notes, mimicking what humans would ordinarily do.

2.6.5 Turn Taking

Turn taking is a most important aspect in dialogues between humans. Unfortunately for language technologists, much of it is based upon non-verbal signals being sent from one participant to another. The subtlety that exists with turn taking based simply upon pausing (such as occurs during telephone conversations) can be clearly seen. Even the slight delays in transmission that exist in long-distance calls can result in one person talking over the other. Some of the literature touches upon the use of earcons such as bells to mark the end of the systems turn ([DISC 1999, 2000] and [Yankelovich *et al.* 1995]) but there is no real agreement upon their effectiveness. As Abbott [2002] points out, when there are no behavioral or visual clues, it is common for parties to lose track of whose turn it is. When this occurs, a conversation rapidly degenerates into chaos. Abbott suggests the insertion of *safe points*, (relatively high-level dialogue navigation menus) which may be returned to by the user allowing them to re-establish where they are and where they want to go. Both Abbott and Ballentine and Morgan [1999] suggest a simple yes/no question can be used to stabilise a dialogue after repeated failures. Yes/no queries are amongst the most robust speech recognition interactions and are most appropriate when attempting to recover ambiguities or user errors. The use of barge-in, that is the person being able to talk over the systems prompt without having to wait for it to finish, can accentuate turn taking problems although its advantages for experienced user are so great that it is justified.

2.6.6 Ellipsis

Mittendorfer *et al.* [2001] makes the point that users may well prefer natural language elements like ellipses. However, he suggests that their inclusion in dialogues requires a deep level of language understanding. My own work (*Section 4.6.6*) indicates that they are not so difficult to model into VoiceXML based dialogues.

2.6. Indirect Speech Acts

A human says “It’s cold in here.” but means “Close the window”. Stone [2000] concentrates entirely on trying to solve (computationally) the interpretation of this type of utterance where the listener has to draw such inferences. Indeed as he points out speakers *rely on* the listener to do so. Stone does provide a computational approach to interpreting such utterances that is based upon the use of modal logic programming and possession of common knowledge by the participants in the conversation.

2.6. Stochastic Variation of Output

The language generated by speech systems tends to be highly predictable. Exactly the same phrases are used again and again. This lacks the naturalness of human speech that often varies both the syntax and vocabulary used each time something is referred to. Glass [1999] explores the use of stochastic language generation to add naturalness to the output. The main drawback to this is the desire for consistency in interfaces by users.

2.6. Adjacency Pairs

McKinlay *et al.* [1993] point out that turn taking can rely upon the semantic contents of the preceding turn. Adjacency pairs are a very simple example of this. These are pairs of utterances which ‘go together’ because the second utterance is made in response to the first. For example the utterance “Do you have the time of day?” or “Good Morning” invites you to take the next turn.

2.6.1 Insertion Sequences

Ramakrishnan *et al.* [2001] considers insertion sequences at length. If one considers dialogues to take place in the form of adjacency pairs, then a break in that pattern constitutes an insertion sequence. Following is an example of an out-of-turn insertion sequence:

- (Question) Machine - What size pizza do you want?
- (Insertion) Person - A sausage one.
- (Pair to Insertion) Machine - OK, a sausage one.
- (Question) Machine – and what size?

They suggest that the VoiceXML form algorithm, specifically handles this type of insertion and, if one keeps grammars active through a form, this is true.

They also touch briefly on two other forms of insertion. The first is for clarification purposes. For example:

- (Question) Machine – What size pizza do you want?
- (Insertion Question) Human – What sizes are available?
- (Insertion Answer) Machine – Small, medium or large.
- (Answer) Human – Large.

The second is also a form of clarification. For example, *Human – Why don't you ask me the questions in topping-crust-size order* touches on making meta-information available to the user; that is, the user is trying to find out what the system can do, and how it behaves generally. In both cases the utterance must be caught, and dealt with by some other mechanism than the standard VoiceXML form algorithm. One such method is to use sub-dialogues that can be invoked by such patterns of words.

2.6.11 Anaphoric References and Referring Expressions Generally

Lappin and Leass [1994] wrote a seminal paper providing an algorithm for resolving anaphora. It relied on a depth of language understanding that is beyond that being achieved in the project's chosen domain. Kennedy and Bogoraev [1996a, 1996b] followed up with a re-implementation of the algorithm that was only based on shallow tagging of the text.

Androutsopoulos *et al.* [1994] also explore the problems anaphora can cause and looked at a confirmatory tactic of the system echoing back the full discourse referent to the user to ensure that pronominal anaphora are correctly resolved. The simple method of keeping a list of discourse referents against which to resolve pronominal anaphora (using gender and number features) is looked at. Indeed, Dahlback [1997] points out that in human-computer dialogues, the anaphor-antecedent relation seems to be of a rather simple kind, with the distance between them normally very small. The Lappin and Leass [1994] approach was much too complicated; a very simple algorithm that basically worked back from the pronoun and selected the first candidate which matched the pronoun in number and person, and that did not violate selectional restrictions, proved quite sufficient.

2.6.12 Temporal Reference Resolution

Reference to times and dates are common in dialogues. Ohrstrom-Sandgren *et al.* [1997] provide a very powerful algorithm for resolving both anaphoric (e.g. “How is Tuesday, January 30th?”, “How about 2?”) and deictic (e.g. “Shall we go next Tuesday?”) temporal

references in dialogues. It was developed (with Defense Department money) for use by the Artwork project [Weibe *et al.*, 1996, 1997] (neither sighted). Weibe *et al.* [1998] goes on to expand and explain the algorithm in great detail. They use the idea of a temporal unit that is able to capture the temporal information supplied in the dialogue. Interesting these units capture periods of time with an instant having the same starting and ending time. They then develop algorithms to resolve these temporal units back to specific times and dates.

2.6.13 Disfluencies

Abbott [2002] points out that disfluency (‘um’, ‘ah’, correction on the fly and abandonment of sentences halfway through) is a feature of human speech. He points out that VoiceXML does not provide us with the power to catch these phenomena. Indeed, some speech engines are tuned to throw away such utterances as ‘um’ leaving us with no data input to act upon in any event. It would appear that the only viable strategy, when using VoiceXML, is to fall back onto a style of dialogue, which procures information item by item, when this problem arises.

2.6.14 Correction of errors

Mishearing and miss saying are a normal part of human conversation. This is no less true in human-machine exchanges. Mankoff [1999] points out that the complexity of error-recovery dialogues (citing Rajicek and Hewitt [1990]) and the amount gained for the effort (citing Frankish, *et al.* [1992]) affects user satisfaction. In addition, work done by Burkirk and LaLomina [1995] indicates that big improvements are required before users even notice a difference in error levels. All of this suggests there is a strong law of diminishing returns in the area of error reduction. (Mankoff’s references have not been sighted). However, Mankoff does suggest a number of options to help with finding and correcting errors. Based upon some internal checking (like spell-checking or matching days of the week to dates), the user can be offered a list of alternates. For example, if a user says “Friday, 27 February” and the system is able to calculate that the 27 is a Wednesday or the closest Friday is 1 March, the system might well say “The 27 February is not a Friday. Do you mean Wednesday, 27 February or Friday, 1 March”. The natural human strategy on error is to pause and correct, often without any clue that an error was present. Rajicek and Hewitt confirm that users prefer to repeat their input at least once before having to choose from a menu. Finally, just as in human conversations (and computer application), an undo/redo facility is really essential.

Spelling words is a normal human method of disambiguating badly heard words. Ballentine and Morgan [1999] suggest using the military alphabet to resolve user input but this whole area is quite difficult. Hilt and Waibel [1996] looked in detail at the recognition of spelled words over the telephone. The main point for us to draw from that paper is that simply trying to recognise spellings with current speech recognition performance is not a good option. Errors can run at anything like one letter in ten unless the speech recogniser is augmented with a suitable language model (in the Hilt and Waibel case the language model was a list of the 15 million names that could be recognised) and some quite powerful AI search techniques to be able to cope with such a large space. Typically, these facilities are not available in a VoiceXML environment.

2.6.15 Confirmations

Confirmations play an important part in human dialogues. Many sources (including Ballentine and Morgan [1999]) strongly recommend avoiding literal confirmations at each stage of the dialogue. They suggest the repetition of the data in the next prompt. For

example, if the system ask the user on what day they wish to travel and the answer is Thursday, the next prompt might be “OK, travelling on Thursday. What time?” Confirmations of strings of digits are particularly problematic but, given that the chance of an error in a string of numbers as a whole is a multiple of the chance of an error on any one digit, it is probably not good practice to ask for strings of digits over four digits long other than by using the keypad [Eisenzopf 2002].

Glass [1999] refers to statistics in Flammia [1998] that concern the human side of human-machine dialogues in the movie domain. He shows that nearly half of users dialogue turns were acknowledgements (e.g. “okay”, “alright”, “uh-huh”). This indicates that even when talking to a machine humans still provided a large volume of confirmatory sounds and it would be useful to reinforce recognition confidence by using them.

2.6.16 Help

Help is a dialogue phenomenon even though we often think of it in the context of computer applications. A special kind of clarification, the provision of help information (‘Help’) allows a participant in a conversation to find out what can legitimately be achieved. There is general agreement that Help should always be available. Clearly, this extended to those who wrote the specification for VoiceXML 2.0; the word ‘help’ is one of the few words that VoiceXML specifies must always be active in the grammar. This means that the system will always recognise ‘help’ when uttered, allowing the programmer to create a specific response. Many of the sources (Ballentine and Morgan [1999], DISC [1999, 2000] and others) recommend that one should let the user know that Help is always available, and how to get it (although by now users may be learning that simply saying *help* is all that is needed). They suggest that Help is provided on as focussed a basis as possible and that there is some mechanism to terminate the Help mode by the user at any time; by this they mean that the user should be able to barge-in over the Help being provided and tell the system to return to the main dialogue without having to wait for the Help to finish. Sharna and Kunins [2002] suggest that a tutorial should also be a characteristic of a good speech application.

2.□ Database Interfacing Issues

2.□.1 Introduction

Much of the work done in this area was carried out in the pre-speech application era when the hope was that (keyboard entry) natural language interfaces to databases would extend structured query language interfaces for non-skilled users. As it turned out, the graphical user interface won the day. However, the work deals directly with natural language and it is still of direct relevance for speech applications. In this project I only look at relational databases. Androutsopoulos *et al.* [1994] provides an excellent review of the state of research in 1994 and I have relied heavily upon it. While interfacing with databases is not considered main stream natural language technology territory by some practitioners, it is interesting to note that Norton *et al.* [1996] considers that a major bottleneck in the development of practical spoken language applications is the interface between the spoken language system and the back-end application.

2.□.2 Response Generation

Androutsopoulos *et al.* [1994] point out that if one wishes to move beyond simply printing the tuples returned from a database, one has to cope with the following phenomena:

- Information contained in databases is very often encoded or abbreviated in some way (e.g. 'Brisbane' may well be included as 'bri').
- Typically, the system has no deep understanding of the questions. If the question is nonsense, it may still be put to the database but no tuples will be returned. The system should explain why the failure occurs rather than say that no results emerged.
- The user's question may contain false presuppositions about the data or the world. In this case the system must correct the presupposition. (e.g. "Is there a 10:30 to Adelaide?" □□ "No" is not the correct answer if there are no flights to Adelaide at all.)
- The user's request may not express literally what s/he wants to know (e.g. "Is there a flight to Athens?" □□ "Yes" is unlikely to be the full information the user wants.)

While the last three problems are quite difficult to solve, one approach to the first is to have a response formatting module to expand information recovered from the database in encoded form automatically (bri □□ Brisbane).

Dale and Reiter [1996] specifically review the role of the Gricean Maxims in the generation of referring expressions from original database type information. Fortunately, the paper indicates that complete economy in expressions is not required. Humans often produce referring expressions with a little redundancy in them but if one takes a task-oriented approach, 'appropriate' referring expressions normally 'fall out of it'. If, when one designs methods to produce referring expressions, one creates an agenda of the characteristics they should have, and then works your system towards them, then one will (more or less) have conformed to the Gricean Maxims.

2.□.3 Database Updates

While most systems restrict themselves to obtaining information for the user (the select structured query language statement), any general solution for database connectivity must also support the other classical structured query language operations (update, delete and insert) [Androutsopoulos *et al.*, 1994 at page 37]. Without such additional functionality it is impossible to write applications that allow the user to conclude transactions or update records.

2.□.4 Meta-□nowledge □uestions

There are times when an ability to ask questions about the nature of the information in the database is useful. Androutsopoulos *et al.* [1994] refers to this type of query as a meta-knowledge question. Empowering the user to ask this kind of question goes some way to solving the three more difficult points referred to at *Section 2.7.2* above.

2.□ The Use Of Tools in the Design of Speech Applications

I reviewed five tools that assist developers to produce speech applications: Audium 3, the CSLU Toolkit, Suede, the Universal Speech Interface and VoiceGenie IDE.

Audium (www.audiumcorp.com) does something very interesting. Using a highly visual interface, (shown at *Figure 5*) it abstracts the dialogue design into the form of a number of states and transitions, allowing one to attach the necessary prompts, grammars and so on to these states and transitions. This fits in nicely with the finite state machine view of dialogues that is prevalent. At the click of a button, it produces the VoiceXML code. Rational Rose and PowerBuilder abstract object-oriented design or database design in much the same way and allow one to write Java or Oracle SQL at the click of a button. Tools that can do this allow one to work at a higher level of abstraction and can assist in improved design. Audium even assists with database connectivity.

The CSLU toolkit (<http://cslu.cse.ogi.edu>) (which is a non-VoiceXML tool) is rather older but goes further in the visual vein. As can be seen from the screen-shot shown at *Figure 6*. The use of the pallet together with a drag and drop technique has proved so user friendly that school children are writing speech applications with it.

Suede (<http://guir.berkeley.edu/projects/suede>) is actually a Wo \square simulation tool but has been included here as the interface shows the characteristic state and transition type layout that seems to fit naturally with dialogues. (The interface is shown at *Figure 7*). It allows one to construct a dialogue using a finite state machine approach and to add the system prompts at each stage. It then runs using text-to-speech to render these prompts to sound and records the user responses.

The Universal Speech Interface ('USI') (www.cs.cmu.edu/~usi/intro.html) (which is an XML, but not a VoiceXML tool) sacrifices natural language for consistency in the interface. The entire idea behind USI is to attempt to do for speech what Palm's Graffiti has done for mobile text entry and Macintosh's universal "look and feel" did for graphical user interfaces. It does not use a highly graphical interface but it writes simple applications very quickly.

VoiceGenie IDE (www.voicegenie.com) is a basic VoiceXML Grammar editor. It provides support for object attributes (which can be most useful given the complex document object model behind VoiceXML), formatting, grammars and debugging. It takes typed input but simulates execution and provides spoken output. The main graphical user interface is shown at *Figure 8*. Such tools are very useful in increasing programmers' productivity and many readers will have used similar tools like the Microsoft Visual Studio for Visual Basic or C++.

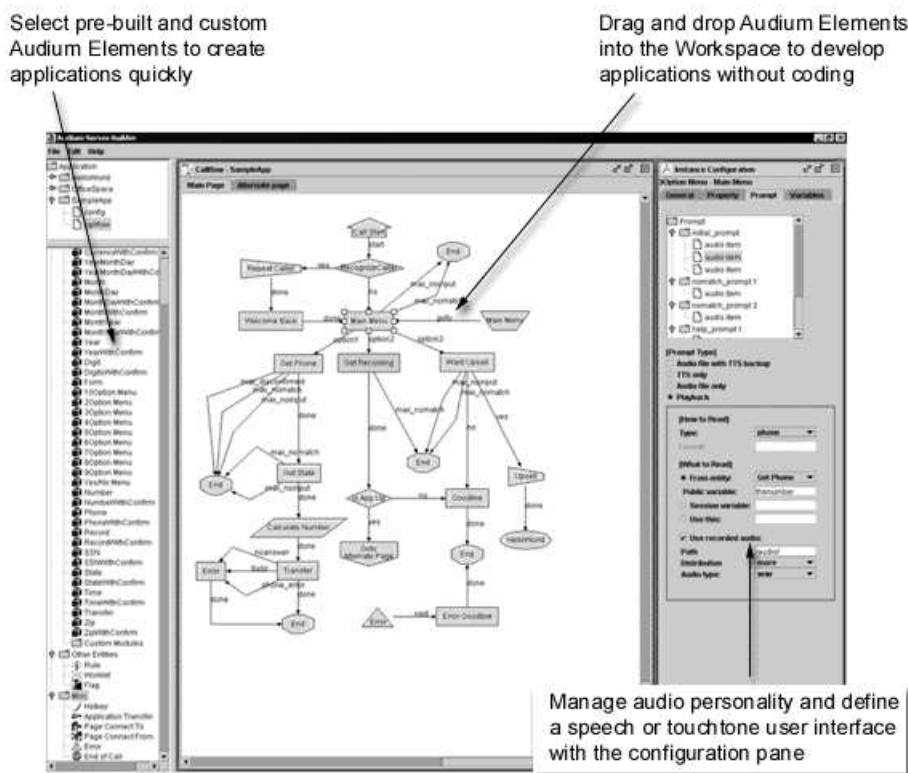


Figure 5 – A screen print of the main Audium interface.

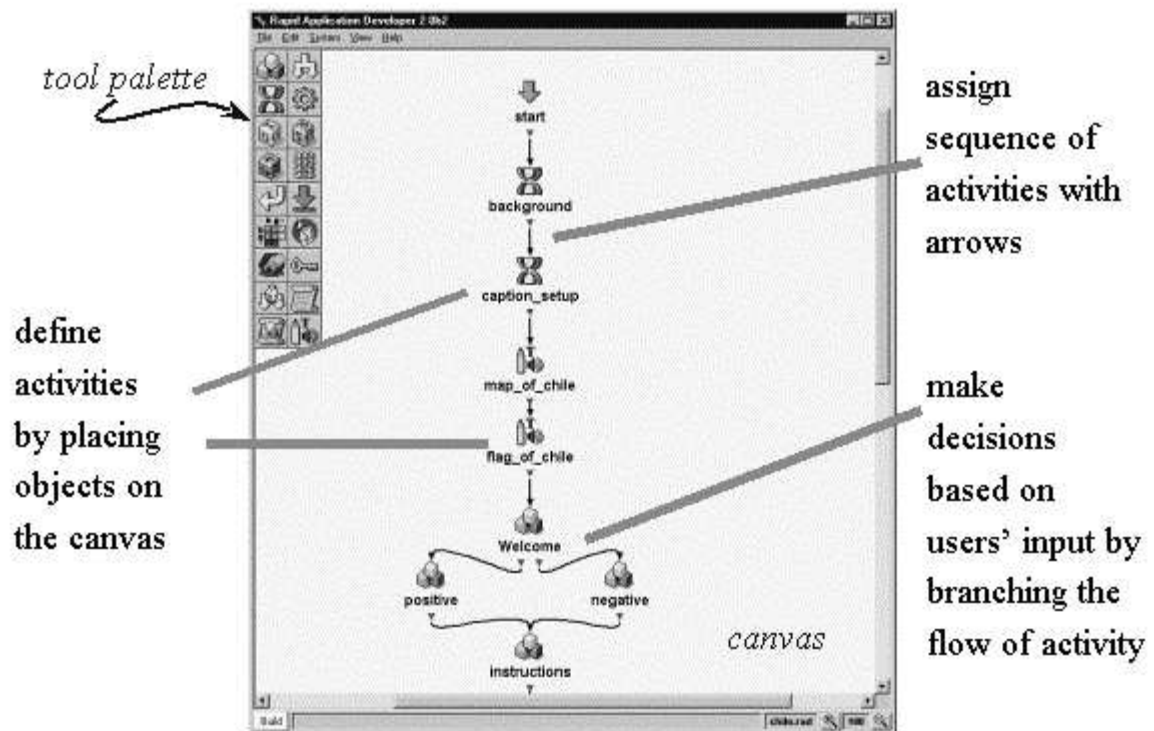


Figure 6 – A screen print of the main CSLU interface

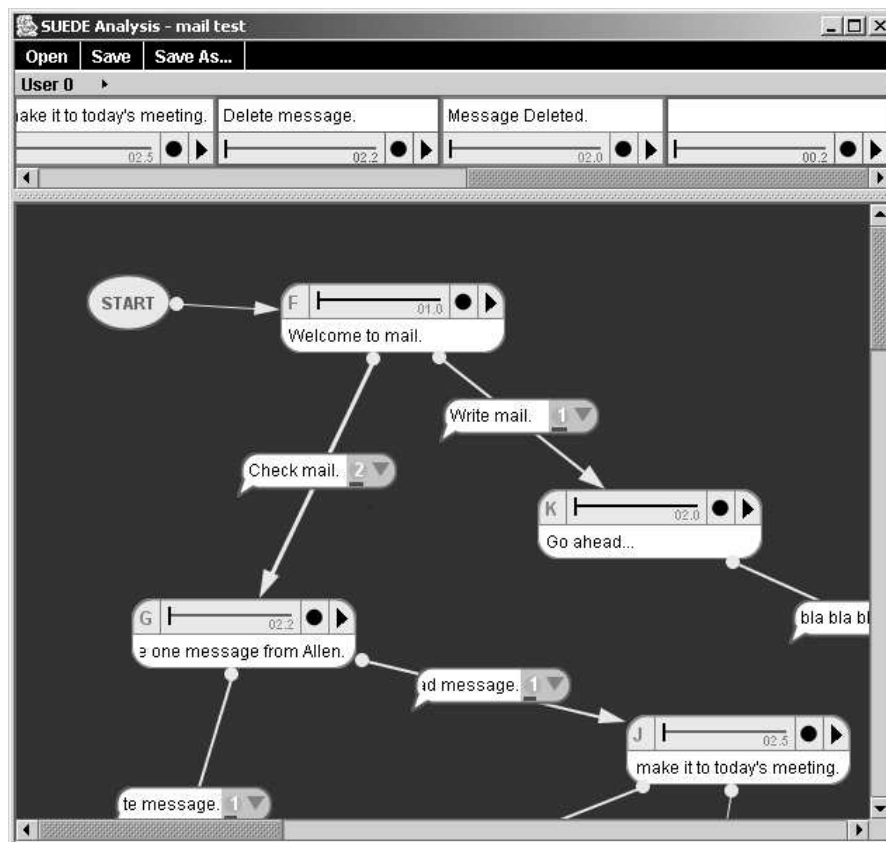


Figure 7 – A screen print of the main Suede interface

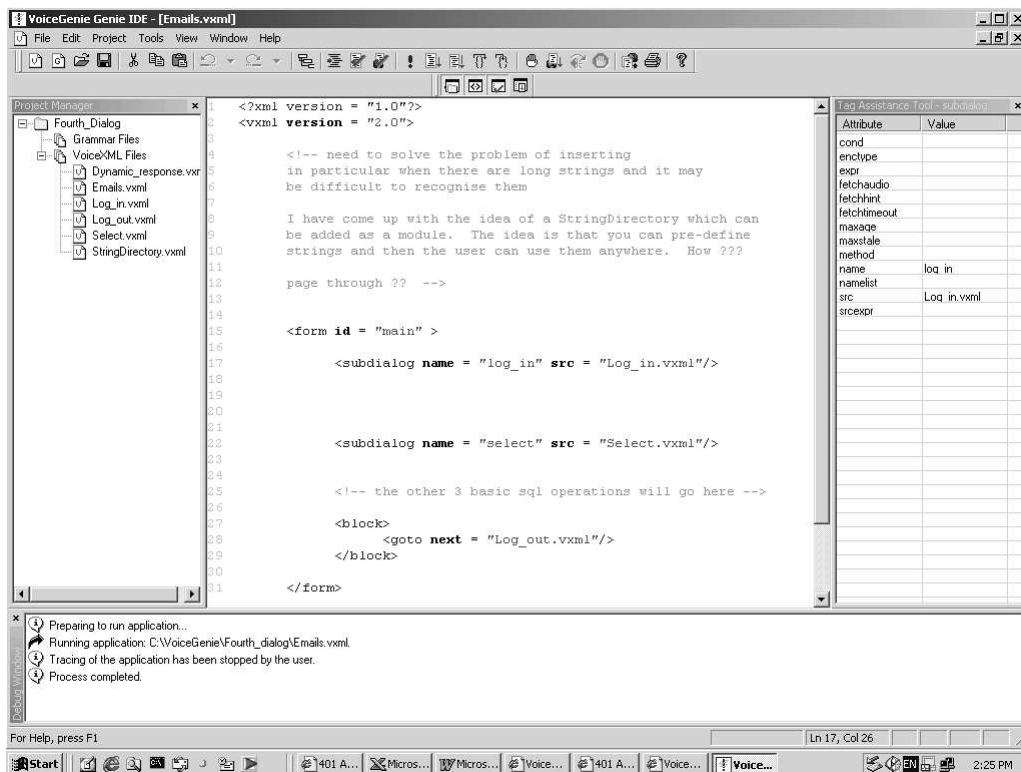


Figure 8 – A screen print of the main VoiceGenie IDE interface

[Kolzer 1999] describes an unfinished tool being developed in the DaimlerChrysler Research Laboratory that could be of considerable relevance to this project. Her work is designed to produce a universal approach for specifying task-oriented dialogues. It uses a number of concepts, one of the more important of which is to abstract dialogues into turns, each of which is characterised as one of a set of dialogue acts (confirm, enquire, etc). This is ideal for representation in a visual form. Increasing amounts of details are then ‘hung’ onto these turns (such as the data types being dealt with *e.g.* string or time). The implementation then creates transitions from each turn to any other turn (save those that have been disallowed by the developer) and checks the consistency of the data being handled. I have made inquiries to see if this work has been progressed but unfortunately at the time of writing no news is yet to hand.

2.□ Conclusion

In this chapter we have looked at applications fielded by industry and academia (*Section 2.2*), at what the literature has to say on good dialogue practice (*Section 2.4*), building speech applications (*Section 2.5*), and dialogue phenomena (*Section 2.6*), and at the use of tools in the design of speech applications (*Section 2.8*). All of this related work leads us up to the point of deciding upon what my application (VoiceDBC) should concentrate.

None of the reviewed tools supports good dialogue practice nor do they attempt to model many of the characteristic dialogue phenomena. Only one provides some help in understanding a database or mapping information into natural language. Interestingly Kolzer [1999] does indicate that she would like to include ‘guidelines of dialogue management’ and cites [Bernsen *et al.* 1998]. I have not sighted that work directly but it does appear to be influential in the DISC [2000, 2001] work, which has a lot to say about tools to enforce good dialogue practice.

It would appear that there is a growing demand for simple well-written speech application [Rosenfeld *et al.* 2001] that can be produced cheaply. So it seemed to me the course was clear. VoiceDBC should provide the user with a tool that can make it easy to build good speech applications. It must support good dialogue practice and it must model relevant dialogue phenomena. It must make it easy to review the data an application has to handle, and to handle that data in practice. If required, it should forgo the ability to handle complex applications in order to achieve these goals.

Chapter 3 – Dialogue Design Patterns

3.1 Introduction

In the last chapter we looked at the existing work related to this project and concluded that I should build a tool (VoiceDBC) that can make it easy to build good speech applications. It must support good dialogue practice and it must model relevant dialogue phenomena. It must make it easy to review the data an application has to handle, and to handle that data in practice. To do so constitutes a large undertaking, probably beyond the ability of one person in one academic year. It was important to be able to simplify the task.

During my research I had come across work on *dialogue-task distance* and I was generally aware of *design patterns*. These two concepts, which are explained further below, allowed me to adopt a radically different approach to the methodology to be used by a developer when employing VoiceDBC in writing a speech application. VoiceDBC would come with templates for various types of speech applications (*e.g.* a timetable template, a take-away menu template, a ticket sale template, an email reader template, an information providing template and so on). Although this approach restricts the usefulness of the tool to those applications that fit a template, it makes it easy to write such an application in minutes. In addition, over time, more templates can be added allowing VoiceDBC to handle most of the application space.

This is a relatively short chapter but is central to how a tool like VoiceDBC can work. In it, we will look at the two concepts introduced above and how they make it possible to use a new paradigm in the development of speech applications. We will then work through the *Plane Timetable Application* to explore the question of implementing dialogue design patterns in practice, and then we will look other applications this dialogue pattern can handle.

3.2 Dialogue Design Patterns

3.2.1 Design Patterns

As Grand [1998] explains, volumes have been written on design patterns, commencing with two books by Alexander [1977, 1979], an architect. Alexander's ideas were taken up by the IT community and Gamma *et al.* [1994] produced a seminal book called *Design Patterns*, which has become something of a bible in object-oriented circles. The essential idea is that a problem can be taxonomised in a particular way and, if one can discern the correct nature of the problem, one can reuse a previous design solution developed for a problem of the same type. I decided to explore this style of thinking as a way to partition the application space amongst different types of applications. I knew that these dialogues all had to map into the one or more of the four basic structured query language operations, and though this would constrain them even more. This should make common patterns all the more likely to find.

The idea of dialogue design patterns did not solve a secondary problem. Some task-oriented dialogues appear to be simple and some very complex. Patterns seemed to capture groups of the simple dialogues but did not seem able to cope usefully with the complex ones. During the research for this project, I came across work on dialogue taxonomy by Dahlback [1997]. Its most important feature, in the context of this project, is to bring out the concept of *dialogue-task distance* with respect to task-oriented dialogues. All dialogues clothe an underlying non-linguistic task and Dahlback observed that there

appears to be a closer connection between task and dialogue in, for example, an advisory dialogue than in an information retrieval dialogue. Dahlback gives this example:

“□ to answer the question of when there are express trains to Stockholm within the next two hours, in most cases there seems to be no need to know why the questioner needs to know the answer.”

The point is, we do not need to understand the underlying non-linguistic task, so that when the distance is greater, we can produce dialogues in which it is not required to understand why users are asking their questions in order to produce useful outcomes. Dialogue-task distance can be used to partition the space occupied by task-oriented speech applications. On the distant side of the partitioning line are applications that do not require any sophisticated feedback in the dialogue (such as timetables) and, on the close side, applications that do (such as medical diagnostic applications). The latter type of application really requires a very sophisticated design function. It cannot be reduced to a pattern, or at least any pattern produced is probably going to be so specific that it proves of little use for any other task-oriented dialogue.

3.2.2 Finding Patterns

At the start of the previous chapter, we looked at the application space and reviewed the academic and commercial applications that have been developed. Many of them cluster in small areas, like flight information, or on and around common themes, like news information of various types. This led me to work on my own taxonomy for dialogues. While it is accepted that taxonomies are essentially subjective methods of classification, the work made it quite clear that there were many common dialogue patterns some with as little difference as substituting ‘bus’ for ‘train’ to turn a train-timetable dialogue into a bus-timetable dialogue. My taxonomy is produced below at *Figure 9*. The groupings used sprang easily to mind as I wrote down the various applications. The idea of viewing them as abstract classes (in the *object-oriented* sense) followed almost immediately, as did the further object-oriented concepts of inheritance and instantiation.

3.2.3 Using Dialogue Design Patterns in Practice

There are at least two approaches to the use of the dialogue design concept in practice. Given the hierarchical nature of my taxonomy, one obvious approach is to start at the top of the taxonomy and work down. On the other hand, as this concept leads to instantiations (real applications), an obvious counterpart is to start with real applications and work upwards.

The top-down approach involves the writing of a cascade of abstract classes that lead gradually to actual instances of speech applications. One might start with an abstract class that is simply a place marker for task-oriented dialogues. It might only carry the attributes of a greeting and a parting comment; both are common to all these types of dialogues. A further abstract class that involves the provision of information might inherit this class. This implies some database connectivity; still at a quite abstract level. A further abstract class might inherit this for timetable inquiries. This class might have prototype methods concerning the traversal of data and attributes that recognised the fact there is somewhere to leave from and somewhere to arrive. It might have methods that recognise that the user will wish to home in on a particular item amongst the data – one flight or one bus. Finally, all of this could be inherited by the actual instantiation of an application. At each level, the classes take on more attributes and methods until finally, an actual speech application is forthcoming.

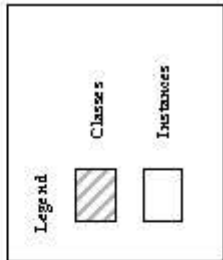
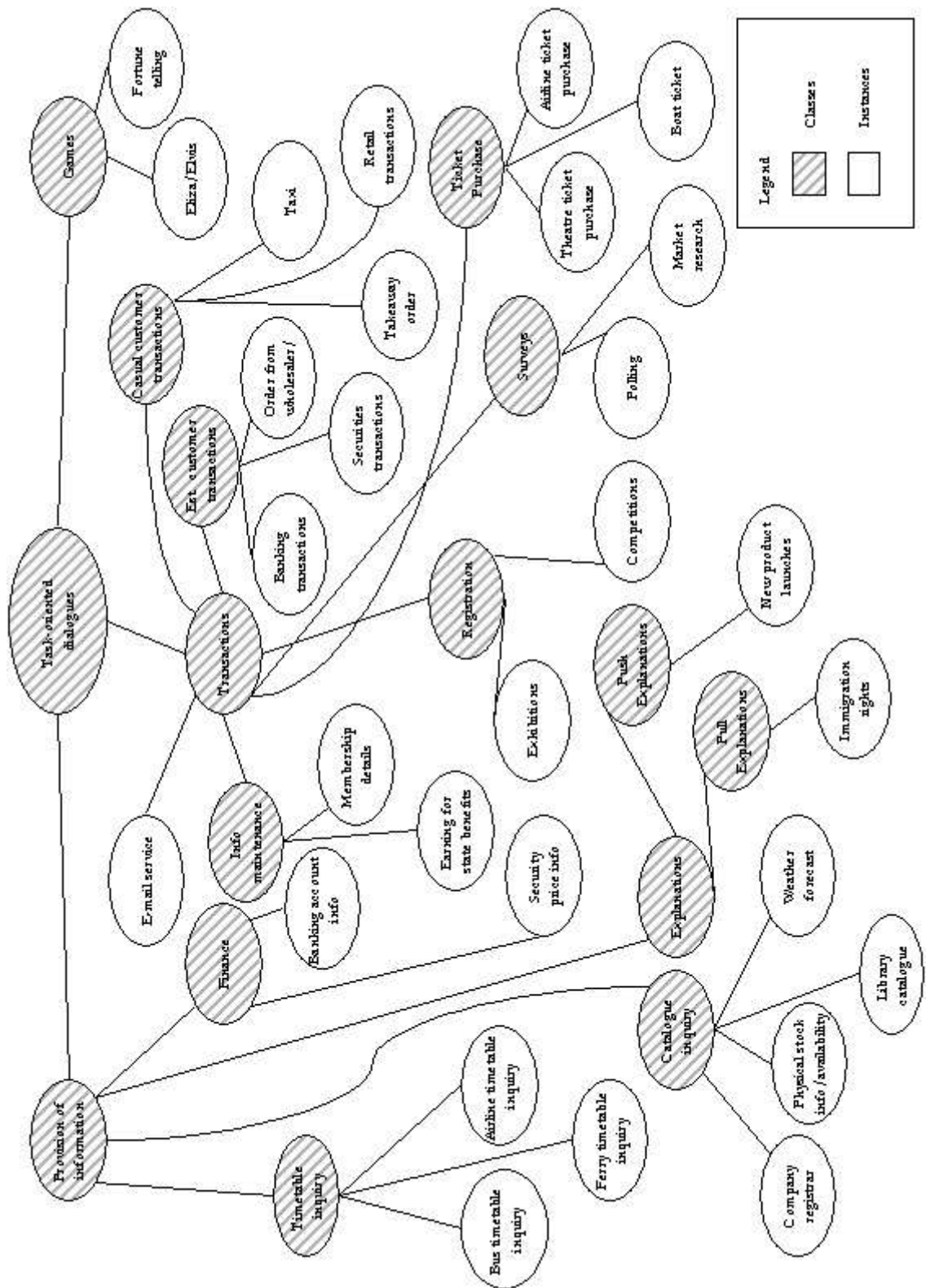


Figure 9 – A dialogue taxonomy

The bottom-up approach involves jumping straight in, writing several instances of speech applications and exploring the opportunities for re-use of the designs, and the components one finds. Given the lack of knowledge about dialogue design patterns, it would have been difficult to complete the taxonomy and abstractions involved in the first approach. In order to gain more knowledge, I chose the second approach, and two instances were implemented:

- A plane timetable application
- A take-away menu ordering application

In order to explore the possibilities for re-use, two further applications were implemented:

- The plane timetable pattern was used to produce an email reader.
- The take-away menu pattern was used to produce a shopping catalogue application.

3.3 Plane Timetable Application

3.3.1 Introduction

The task to be discharged by this dialogue is to allow a user to phone in, and find out when there are flights from one capital city to another. In forming a design pattern I bore in mind that:

- The system should be able to offer close flights, if the exact one a user wants does not exist.
- It is more efficient to access a database once, and bring back a larger set of information than is required, than to undertake repeated accesses as one homes in on the flight time that is required.

3.3.2 The Basic Dialogue Design Pattern

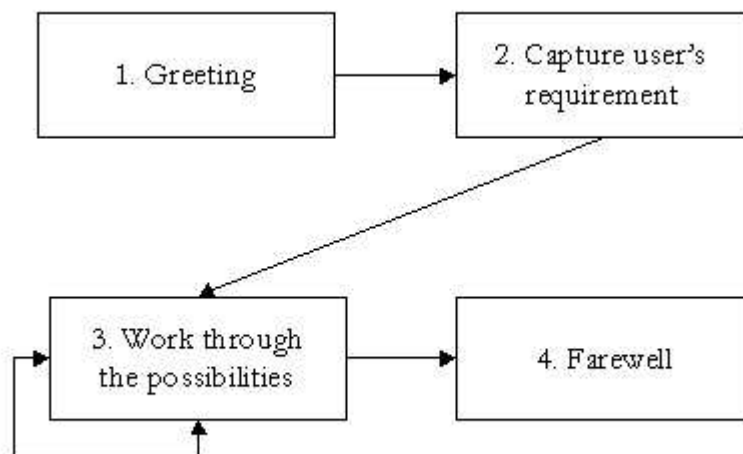


Figure 10 – The basic dialogue design pattern for a plane timetable application

Clearly, the design pattern employs the ubiquitous ‘hallo’ and ‘goodbye’ of human dialogues. The first task following the greeting is at *Stage 2* – to capture the user’s requirements. To do this, the developer must make some decisions. To ask for a precise requirement has a number of problems. It accentuates the possibility of a useless reply

(e.g. question - 'Is there a flight to Darwin from Sydney at 10:30?', answer - 'No' when there happens to be one at 10:35 or indeed there are, in fact, no direct flights to Darwin from Sydney). The pattern I adopted was to ask for the minimum set of information needed to procure a set of data from the database that is likely to contain the case the user is looking for. In this case that is:

- Where are you going from?
- Where are you going to?
- What day do you want to travel on?

A further choice is required, if the set returned is large, what criteria will be used to focus in on the users requirement? In different domains, the factor will be different. In this domain, one might choose the departure time.

At *Stage 3*, the system has procured the data set and works through it with the user. There is work [Walker *et al.* 1998] that indicates users tend to be happier with a simple 'start at the beginning and work through the data' approach for small data sets and if the set is under seven that is what occurs. If the data set is over seven items, the application should ask for clarification and range over the data until it find the closest match and then start offering the data from that point. The user can loop either way on the data and all the normal functionality of back, forward, first and last, so common on graphical user interfaces is incorporated in the design. Indeed, if users become familiar with that approach, it will be considerably easier to use dialogue design patterns across different domains than otherwise; the language employed for this navigation task could be static and not have to change from 'Which city do you wish to depart from?', to 'Which suburb do you wish to depart from?' as one moved from planes to buses.

Finally, at *Stage 4*, the system says goodbye.

3.3.3 Using this dialogue design pattern elsewhere

It is easy to see how this dialogue design can be used for planes, buses, ferries and trains but it is not so easy to see how it can be used for email reading. However, if we just think of emails as being some text divided into a few important fields, such as the time of transmission, the name of the sender, the subject matter of the email and its text, it is easy to see how the four steps apply directly.

1. Greeting
2. Capture users requirement - does s□he want all the emails, the ones not previously read, the important ones?
3. Work through the possibilities - if the set is large use some focus to manage it otherwise start reading the first one.
4. Farewell

3.4 Conclusions

In this chapter we have looked at how the concept of dialogue-task distance helped me divide the application space into those simpler applications which a tools such as VoiceDBC could handle. We also looked at the pivotal role the concept of dialogue design patterns plays in allowing VoiceDBC to build speech applications. Without being able to categorise applications in some way it would be impossible make sense of the idea of templates that can be used across similar applications. We also looked at the use of dialogue design patterns in one particular application - *The Plane Timetable Application* -

and how that pattern could be re-used elsewhere. In the next chapter we will review the actual implementation including what features were incorporated, and what was left out.

Chapter 4 – The Implementation

4.1 Introduction

This chapter concentrates on the actual implementation of VoiceDBC. First, we look at the general architecture (*Section 4.2*). Then we look at implementing design patterns (*Section 4.3*). Next, we work through the incorporation of good dialogue practice (*Section 4.4*). This is followed by a section on some of the general problems that the current state of speech recognition give rise to (*Section 4.5*). Finally, we look at dialogue phenomena (*Section 4.6*) and database interfacing (*Section 4.7*). During the chapter, I cover features I have included and also those that have been left out of VoiceDBC. Omissions arose for two reasons:

- They concerned matters (areas of good dialogue practice or characteristic dialogue phenomena) that did not arise in the speech applications I wrote.
- Having solved them in principal, they involved a considerable amount of coding when time (which was at a premium) could be more interestingly spent exploring new and challenging phenomena like handling long lists of data in dialogues.

I will flag these features as accurately as I can, as they constitute a fertile ground for further work. The code written to implement VoiceDBC is contained on the CD that accompanies this thesis.

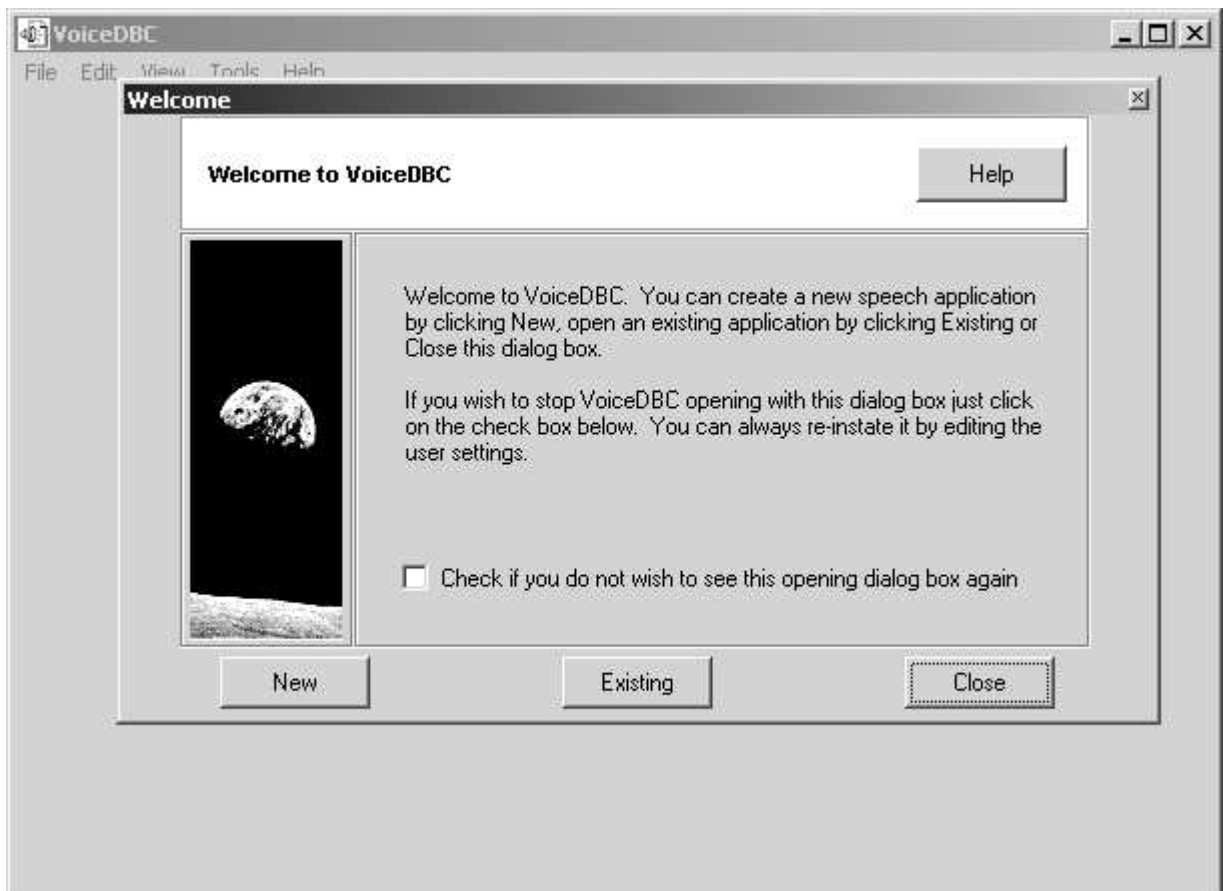


Figure 11 – VoiceDBC, the opening screen

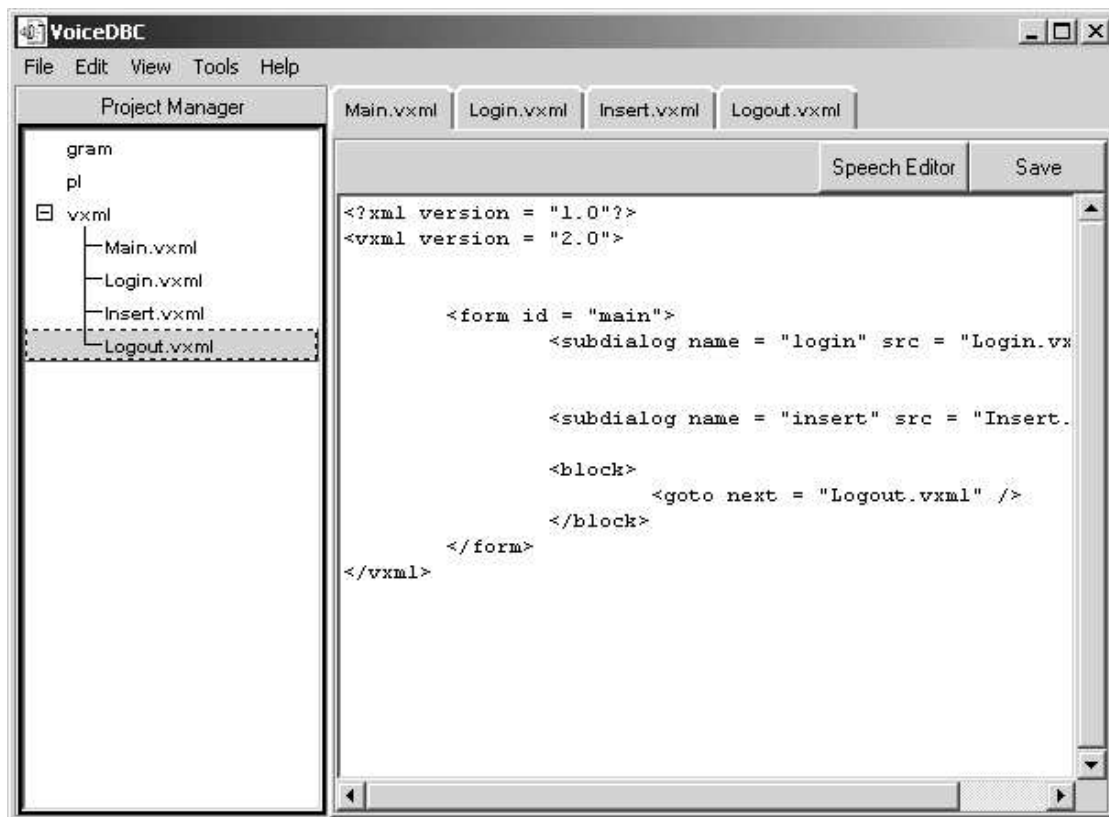


Figure 12 – VoiceDBC revealing its multi-document text editor

4.2 The General Architecture

The writing of speech applications, in our domain, involves the outputting of a number of text files – most are in VoiceXML but some are grammar files and some Perl files destined for the cgi-bin. With this end in mind, VoiceDBC contains a text editor (see Figure 12) and all these output files for a particular project are kept in a directory with the same name as the project (e.g. *take away*). All the projects are kept in a sub-directory called *projects* so the path to any particular set of documents might be `projects/take away`. Deployment of these final files is left up to the user.

The templates are kept in a similar directory structure, but the facility has been added to separate them by persona if this becomes desirable at some later date.

Task-oriented dialogues with a large dialogue-task distance lend themselves to abstraction into two parts: *data* and *templates*. The data is that part associated with getting and handling the information in the database. The templates are patterns of the dialogues as reflected in partially abstracted project documents. Perl provides a number of modules that can help us handle these aspects and I made extensive use of three modules CGI, DBI and HTML::Template.

There are a number of approaches for separating interfaces from the code (see Figure 13). HTML::Template has proved a very popular one. Certainly it makes it easy to start working with instances of the speech application and then to abstract those parts which may change based upon the nature of the data to be processed. Given that I had decided to learn more about dialogue design patterns by adopting this approach, it suited my purposes admirably and allowed me to handle any difficult text data constructs in the main Perl code.

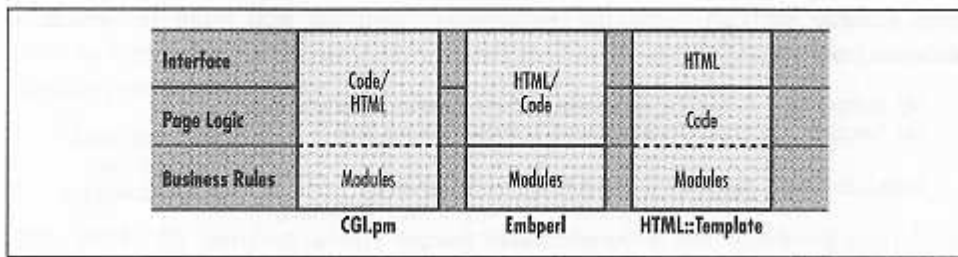


Figure 13 – Approaches for separating interfaces from code⁵

4.3 Implementing the Design Patterns

First, I implement dialogue design patterns by writing them using traditional methods. In my case, I made extensive use of the VoiceGenie IDE. Those parts of them that are dynamic (that is, which change given the underlying nature of the data) are turned into variables that can be filled from the main Perl code. Design Patterns therefore end up being captured in two places – the basic interface in a template and the dynamic parts in the main Perl code. After the first application was implemented it became obvious that some parts of these dialogues are really universal, and some common templates for handling opening and closing greeting were created.

At first, two instances were written:

- A plane timetable inquiry service.
- A (large menu) take away service.

In order to demonstrate that the patterns found in these applications could be used in seemingly quite different domains, two further applications were written using the original patterns:

- An email reader
- A catalogue sales application.

I have characterised these patterns ‘locate and review’ and ‘offer and select’. Writing the initial instance of the pattern saw all the usual problems which are associated with coding but the ‘clone’ was created mostly using a ‘find and replace’ technique common to text editors. Of course, the changes in data could all be handled dynamically.

4.4 Incorporating Good Dialogue Practice

4.4.1 Introduction

It is probably useful to restate the definition of good dialogue practice introduced at the beginning of this thesis:

Those strategies and tactics which, when adopted in the designing of a dialogue, produce speech applications that are effective, easy to use and not prone to error.

These strategies and tactics come in two forms:

- Rules that apply across an application, such as the requirements for consistency of manner and reactions, (e.g. don’t switch from a professional urban style of output to broad Australian bush style in mid-dialogue or when the system does not hear a

⁵ From page 141, Guelich *et al.* (2000). *CGI Programming with Perl*. O’Reilly, CA.

user's utterance at one point of the dialogue, use the same words in response as are used elsewhere else for this problem).

- Rules that apply to particular utterances such as these from Balentine and Morgan [1999]:
 - When prompting for Yes/No answers, use the interrogative form (*e.g.* "Is this correct?").
 - Include the verb and only use the imperative form for error-recovery (*e.g.* "Please answer "Yes" or "No".")

I relied heavily on Balentine and Morgan in this latter area. With a tool such as VoiceDBC (which produces finished code) support for good dialogue practice must be designed in rather than offered as guidance to users. For the balance of this section we will look at how this is achieved. It should be pointed out that a checklist for this area was not prepared, and this would constitute a useful area for future work.

4.4.2 Forms in VoiceXML

VoiceXML offers two principal types of user interaction, the *menu*, and the *form*. The menu is an analogue of the visual menu, where a list of options is offered to the user, and a choice made. The form allows for a subtler interaction. Initially, a form allows a user to make an utterance, which can contain one or more of the pieces of information the system is looking for (*e.g.* this sentence contains two of the three pieces of information a system might be looking for: 'Can I fly to Sydney on Tuesday?'); subsequently the form allows the system to take the initiative in obtaining any other information it requires (*e.g.* the final piece of information being looked for might be 'Where are you departing from?'). The form algorithm runs through any still unfilled fields one-by-one until the required data is collected. It endeavours to free the dialogue from a slavish question-answer format, and shortens the time required for an experienced user (one who knows what information is required at that point) to navigate the dialogue.

4.4.3 Form Specification

Many sources consider the use of mixed initiative dialogue to be good dialogue practice ([Balentine and Morgan 1999], [DISC 1999, 2000], [Glass 1999] and others). It is useful to consider the basic nature of a form before elaborating it with all the various prompts, re-prompts and help that will ultimately be required. At its most basic, a form consists of some VoiceXML code (an example is shown at *Figure 14*), supported by a grammar which helps the underlying speech engine recognise user utterances (an example is shown at *Figure 15*).

Good dialogue practice asked us to go well beyond these simple examples as will be explained in *Section 4.4.4* below. A VoiceXML form allows us to count the number of times we have to prompt and re-prompt and make the system say different things on each occasion. It allows us to create more meaningful utterances than the usually rather banal platform defaults for non-recognition and no-input. It allows us to offer help. Some of these actions (*e.g.* no-input) can be well handled at the application level but many are directly related to the *field* within the form where the bulk of the work is performed in terms of human-machine interaction.

4.4.4 Fields

Within the form, it is the field that actually tries to capture data, and has to produce the most sophisticated responses to comply with good dialogue practice. Prompts, no-

match and help must be covered for each field and must be relevant in their context. None of the literature gives any guidance as to the number of different utterances

```

<?xml version = "1.0"?>
<vxml version = "2.0">
  <form>
    <grammar version = "application/x-abnf" src =
      "reply.gram" />
    <initial name = "required_info">
      <prompt>
        can i help you with your flight
        requirements?
      </prompt>
    </initial>
    <field name = "from_city">
      <prompt>
        what city do you want to fly from?
      </prompt>
    </field>
    <field name = "to_city">
      <prompt>
        what city do you want to fly to?
      </prompt>
    </field>
  </form>
</vxml>

```

Figure 14 – A Mixed Initiative Form

required but a user will require a more significant type of help (an operator or returning to some safe-point) if the system fails to cope with any utterance at any particular point more than a few times. Many VoiceXML platforms simply give up after three attempts (the alternate seems to be to loop forever), and so a sensible approach might be to offer two or three styles of utterance followed up by a transfer to an operator. These utterances should themselves be designed to promote good dialogue practice as follows:

- prompt 1 A prompt made up of any brief instructions followed by the call to action.
- prompt 2 The call to action.
- prompt 3 The call to action.
- no-match 1 Direct but terse guidance as to the correct form of reply, *e.g.* “Say the name of the city.?, ‘Please just answer yes or no?’.
- no-match 2 A fuller explanation of what the system can understand, *e.g.* “In relation to your next utterance I am programmed to recognise the following words, x, y and z.”
- no-match 3 “If you wish to be talk to a human just say ‘Operator’.” Or the calling of a spelling sub-dialogue to disambiguate.
- help 1 As per no-match 2
- help 2 As per no-match 3.

It might seem strange to suggest that prompt 2 and 3 are the same but Ćajicek and Hewitt [1990] confirm that users like the opportunity to repeat an answer prior to going into more complex recovery routines. In addition, the mirroring of the no-match response in help assumes that a person wanting help will be facing the same problems as one who is saying the wrong things, therefore, individual crafting of additional help

responses is unnecessary. There is considerable guidance, particularly in Balentine and Morgan, as to the correct tense and vocabulary to use in any particular case and it would not be sensible to basically repeat large chunks of the literature directly (readers interested in the detail can refer to Balentine & Morgan [1999], *Chapters 2 and 3* in particular). To incorporate this material into VoiceDBC requires one to write the templates with one eye on the guidance provided. Rigorous compliance should be enforced with checklists. I wish to flag this elaboration of the field element. Providing multiple responses at every stage of a dialogue, expands the code to be written in an exponential fashion without making any great contribution to our understanding of good dialogue practice. It is one of the items that has to await a period when deadlines are not an issue.

4.4.5 Grammars

Generally, understanding how a field works is unclear without considering its grammar. This defines the utterances that will be recognised at that point and can allocate semantic values to those utterances or simply accept the recognised utterance as the value to be used. There are a number of problems in deciding how to produce the grammars that enable recognition of the user's utterances. Some considerations are simple. Generally, we should choose multi-syllable words or phrases, and avoid noisy words (containing unvoiced fricatives, 'P' and 's') and low-energy words (containing phonemes such as the

```
#ABNF 1.0 UTF-8;

language en;
mode voice;
root $reply;

/*****

    reply:examples

    please can i fly from sydney to darwin
    can i fly to darwin from sydney

*****/

public $reply =
    [$please] [$can] ([$from_city | $to_city] | [ $to_city |
    $from_city] );

$please = please;
$can = can i fly;
$from_city = from sydney {sydney} | from darwin {darwin};
$to_city = to sydney {sydney} | to darwin {darwin};
```

Figure 15 – An ABNF Grammar

nasal 'm' and 'n'). In addition, we should avoid word pairs that have excessive syllable sharing. All of this was dealt with in greater detail at *Section 2.4.1* at *Recognisability*. However, an issue of much greater complexity is how far should one go in trying to recognise a wide range of user utterances. Here, there are two forces at work:

- the desire to accept arbitrary language inputs; and,
- the limitations of speech recognition.

There is a strong desire to be fully natural in the language a user can employ. After all, this is natural language technology and those who decide to spend money on new systems like to be persuaded that users will be able to say pretty much anything and the system will recognise it. However, speech recognition still has many limitations. Most systems use defined grammars. Key word spotting is an alternate approach. Under this method one throws away intervening words and catches key words. For example, one might look for city names in a string of utterances and throw away the context. However, this has its own problems as without the context, it is often difficult to extract semantic information and, in any event, key word spotting is not always supported on current VoiceXML platforms.

Unless one specifies everything a person may say in a grammar, nothing will be recognised at all. This subject is dealt with further below at *Section 4.5* and Fraser's [1997] comments on dialogue design (he characterised three types of approach to designing dialogues - design by intuition, design by observation and design by simulation, see *Section 2.5.2.1*) apply equally to the design of grammars. How can one choose the correct responses to recognise, by intuition, observation or simulation? One requires perfect matches if the speech engine is to recognise. If one encourages a wide range of user responses, what is to stop the user adding on an extra word? In itself that is quite enough to make recognition fail and speech applications that are prone to fail are contrary to good dialogue practice. One alternate, the one adopted in VoiceDBC, is to encourage a terse form of exchange from the very beginning. This is designed to restrict user utterances to only those necessary to convey the information required and is in accordance with good dialogue practice at this time. Most of the literature agrees on this point even though most of the commercial players like to put the opposite view forward. Ballentine and Morgan goes even further pointing out, at page 214, that:

Sooner or later, colloquial and casual user speech causes problems. Such behaviors should therefore be negatively reinforced□ .

[For example:] App: *Is that correct?*
 User: "You bet it is."
 App: *For clarity, please reply with a simple □es□ or □No□.*

4.4.6 Blocks

Blocks are sections of VoiceXML code that are executed by the VoiceXML interpreter. Often, they are used to contain logic (ECMAScript or if-else constructs) but they can contain audio, and such utterances must also conform to the requirements for prompts in fields. Normally, such audio does not require any response but provides the user with information or guidance.

4.4.□ Application Wide Rules

Application wide rules tend to be designed to promote consistency. We have already looked at them at *Section 2.3.1* under *Conventions*. The use of style guides (and checking for compliance with them) assists in creating consistency. This can be taken as far as the creation of persona. An early version of VoiceDBC tried to grapple with the issues of persona. However, we have already looked at the question of mimicking in dialogues (*Section 6.1*). While persona may have their uses, one can imagine such personae as 'Crocodile Dundee' or even 'Wog-Boy'⁶ being popular. While Crocodile Dundee's

⁶ For any overseas readers, post war immigration into Australia turned Melbourne into the second largest Greek city in the world. Wog-Boy is a theatrical characterisation of a young second□third generation male immigrant from that part of the world. His accent and vocabulary are quite outside 'standard Australian'.

accent would probably not cause a speech recogniser trained on an Australian corpus many problems, his natural choice of vocabulary is unlikely to be modeled (or indeed known) to a typical speech application developer. Wog-Boy fails on both counts. Given the limited gains and possible drawbacks of going so far as a persona, VoiceDBC did not pursue this avenue. It relies on a single style that tries to model itself on ‘polite professional Australian English’. However, a style guide has not yet been prepared, and this would be a useful area for future work.

4.5 Coping with the Unrecognisable

4.5.1 Introduction

Why should we be interested in handling utterances that cannot be recognised? One aspect of good dialogue practice is to produce speech applications that are not prone to error. Recognition errors are the main problem area. Indeed there are classes of user utterances we know will arise, but which cannot ever be recognised given the current state of speech recognition. What are these and how do they arise?

Current speech recognisers cannot decide upon the meaning of sounds, with any degree of reliability, without some form of language model to assist them. These models are often defined as grammars. As sounds arrive at the system they are first analysed using acoustic models (based on the phonemes that make up the spoken language). The language models are then used to help decide what words the ‘recognised’ sounds make up. As previously pointed out speech recognisers find it easier to recognise ‘speak louder’ than ‘louder’ and simple words like ‘six’ often cause problems [Abbott 2002]. This matching process is the reason behind this. There is more data in the longer utterance and more chance for the essentially probabilistic speech recognition to work. A simple grammar might be:

[Can □ Could] I fly to [Darwin □ Sydney]

It covers such utterances as “Can I fly to Darwin?” and “Could I fly to Sydney?”. It is clearly restrictive and would remain so even if every State Capital was added together with ‘go’ and ‘travel’ as alternate verbs, ‘we’ as an alternate pronoun, and any number of alternate openings like ‘I want’, ‘we want’ and ‘please’.

How can we achieve a larger vocabulary that can be recognised; an open vocabulary? One problem is that the more choices, the more complex it is simply to match the utterance with the possibilities. Some of this arises from the fact that the search methods are based on classical AI solutions and tend to blow out (computationally) so that the requirement that speech applications operate in real-time can be affected. This is the same sort of problem faced with computers playing chess. If you are going to search a space fully, given that the space expands at some sort of exponentially function to its depth, it is easy to run out of time or memory. Some of this arises from the fact that speech recognition is probabilistically based and the more choices, the less chance of hitting the right choice. However, there is an alternate to the simple type of grammar shown above. To write a grammar for the whole language is generally accepted to be unachievable (at this time) but a statistical approach can be taken. A large corpus of the language can be studied and the chance of a word appearing after n others can be calculated. These n -grams (as they are called) can be used to predict the words to be recognised, and can substitute for a written grammar. By this method a (more) open vocabulary can be recognised, but in a way this simply begs the question. To understand arbitrary (but grammatically correct) strings of words, we have to parse them in an exercise very similar to defining a grammar for a language model. If we accept that we

cannot do the former, how can we do the latter? This problem is exacerbated by the fact that spoken English is very much less grammatical than written English.

In some ways, the understanding of arbitrary language inputs is the Holy Grail of language technology. To tease out the semantics from utterances is the natural destination of all this work, but for the moment, many practitioners believe this is an AI complete problem, and will have to wait until all the problems to be solved in AI are solved. Without accepting that proposition, it is certainly true that at this time, we must look to shallower ways of acquiring our information. To see what can be of use, first let us look at the sorts of arbitrary information we are likely to be presented with in our limited task-oriented domain. This is mainly of the nature of proper names, addresses and email messages.

4.5.2 Proper names

Proper names are a continuing problem. Providing the user is not on a silent number, a reverse look up of both name, and address is possible, and no self-respecting voice gateway is likely to be without an API that allows the writers of speech applications to obtain such information. However, that is not open to us. In any event, reverse look up will only offer the possibility of asking a yes/no question, and for those who are calling from silent numbers, or away from home, the problem persists.

There are approaches that might be pursued involving large databases of common names used as language models, but they are not very practical with VoiceXML. If (at this time) one is forced to accept names over the phone, it will have to be by using the standard human solution - 'Could you spell that for me'. VoiceDBC should support this dialogue phenomenon, and some of the coding has been undertaken. However, given current quality of speech recognition, one is almost guaranteed to have to fall back onto some SMS type of keypad input, and this work has been flagged for the future.

In practice, most applications can be expected to rely upon some sort of pre-registration, probably over the visual Internet, and the use of simple four digit pin numbers to identify users.

4.5.3 Addresses

With addresses, the solutions are similar to proper names, but the idea of spelling out an address is probably too tedious to ever be implemented in practice. Capturing the user's utterance in a wav file and its subsequent interpretation by a human, is probably the only viable option.

4.5.4 Email

With emails, the input is quite arbitrary and, one is forced to assume the content confidential, so that human interpretation is not viable. The only option would appear to be capturing the user's utterance in a wave file. When attached to an email message that is sent in reply, this method would normally fire up a wav player on the recipient's computer allowing them to listen to it. There are some size issues with wav files but I understand that there are alternates that are more compressible, but still common enough to fire up when clicked on as an email attachment.

4.6 Incorporating the Characteristic Dialogue Phenomena

4.6.1 Introduction

The ground covered in this section looks at the various dialogue phenomena that the research revealed, and considers their incorporation into VoiceDBC. Two of the

headings that appeared in *Chapter 2*, when we looked at dialogue phenomena in the *Related Work* have been omitted as they have been dealt with quite fully earlier in this chapter. These sections are *Mimicking Style* and *Stochastic Variation of Output*.

4.6.2 Talking to our Audience

This phenomenon has two aspects. First, users bring with them a body of knowledge. Obviously a designer should try to model the system utterances to reflect this. In designing utterances for a system such as VoiceDBC, I was compelled to adopt a ‘man on the Clapham Omnibus’ approach. The nature of the ultimate user is not known at the time a dialogue design is developed. However, there is a second aspect, concerned with how experienced a user is in interacting with speech applications in general, and with the speech application s/he is using in particular. In this aspect, much can be done to tune the application to the user’s experience.

The first essential is that a database be kept that records significant system-user interactions. Most applications will involve some sort of log-in procedure, and, even those that do not, may be able to capture the user’s phone number, and base a history on that identifying fact. Clearly, users with experience do not require the detail or length of prompts that novices do. There should be at least two settings (novice and expert), and prompts should be tuned to reflect this. VoiceXML allows conditions to be applied to most items, and it is relatively easy to have:

```
□ prompt cond□"novice"□ You log-in with a four digit pin number that was provided
at the time you registered for this service. Tell me your pin number now□ .
```

```
□ prompt cond□"expert"□ What is your pin?
```

The classification of new users as novices is easy, but what criteria might be used to re-classify them as expert? Perhaps the simplest method is for the system to ask the novice user, after any period of frequent use, if they want to be re-classified, and then act on the answer. The system should tell them they can revert to novice at any time just by saying ‘novice’. This aspect is flagged for future work.

4.6.3 Linearity

In order to cope with the deficiencies in human memory, human dialogues allow for exchanges to remind a participant of what has already been said. Speech applications should provide for a Repeat prompt, and this can be achieved by a piece of code along the following lines in every field (having enabled the grammar to recognise ‘repeat’):

```
□ filled mode□"any"□
  □ if cond□"repeat"□
    □ script□
      [field name] □ undefined;
    □□script□
    □ goto nextitem □ [field name]□
  □□if□
□□filled□
```

This will allow the user to go back to the prompt just spoken, and, when taken together with an appropriate use of confirmations, goes some way towards solving this problem. Having a form writing module in VoiceDBC could considerably ease the problems of the considerable work created by the continual elaboration of form elements. Both a repeat function, and a form writing module are flagged for future work.

4.6.4 Time Out

Speech applications should allow users to tell them (at any time) to stand by, and they should also allow users (at any time) to ask them to speak more slowly (such slowing down should probably continue to the end of the current system utterance). The former desirable objective may not be appreciated by call-centres, and the like, who desire to minimise line usage, but is certainly consistent with maximising user's satisfaction. Both features are flagged for future work.

4.6.5 Turn Taking

VoiceDBC has to cope with this phenomenon without any of the visual clues given between humans to indicate whose turn it is. Of course, the ending of prompts with a clear call to action (*e.g.* 'What day do you want to fly on?') does encourage proper turn taking and this should be adopted as good dialogue practice. In addition, adopting the policy of frequently using yes/no questions in the dialogue, tends to stabilise, it but in practice, VoiceDBC has to provide ways for the user who is lost, to regain a proper footing.

The most practical way for this to be achieved lies in the elaboration of the prompt, no-input, no-match, and help utterances, and ensuring that they do give information on the location, and role of the current turn in the context of the dialogue as a whole. In addition, in complex dialogues, a facility to go back to the start of the current section may also help, although all the dialogues I have been working with are too simple to require this.

In the event of total failure, nothing should stop the user being able to call upon human assistance if required. To achieve this, I can exploited the fact that the grammars of a VoiceXML root documents are active even when the user is in other application documents, so that the user can always interact with forms, links, and menus specified in them. By providing a form which allows connection to a human upon the use of the word 'Operator' (which is the [ETSI 2002] recommended utterance) in the root document, and ensuring that users know it is available, this lifeline can always be at hand.

4.6.6 Ellipses

This phenomenon should be borne in mind when writing prompts (or perhaps more accurately re-prompts), and when designing grammars. That is, grammars should allow the absence of those parts of sentences likely to be dropped by humans.

4.6.7 Indirect Speech Acts

The only method available for us to cope with indirect speech acts lies in the grammars we write. Any computational methods are quite beyond the processing available to us. The designer must decide if 'Can I fly to Darwin?' really requires a yes/no answer in the context of the dialogue, and, if it really requires 'There are 5 flights, the first one leaves at 10', they must write the speech application appropriately.

4.6.8 Adjacency Pairs

We can exploit the fact that turn taking can be promoted by the semantic content of the preceding turn. As explained in *Section 4.4.4* above, ending prompts with a clear call to action (*e.g.* 'What day do you want to fly on?') encourages proper turn taking. Clearly we must exploit this phenomenon (*e.g.* questions and answers, greetings and responses) fully in the design of system utterances.

4.6. □ Insertion sequences

As Ramakrishnan *et al.* [2001] points out, the VoiceXML form algorithm provides some support for the ‘out of order’ type of insertion. However, we are still left with the clarification and meta-information type of insertion. Both of these tend to be Wh-sentences (‘How □ ’, ‘Why □ .’, ‘Where □ .’, etc), and it is possible to catch some of them by writing a suitable grammar. However, normally there is little one can do to answer them as their arbitrary nature makes it difficult to design stock responses. The normal response must therefore be to drive the human back to the responses the system is geared up to accept. To do this, one can simply leave Wh-sentences as no-match, and generate the normal system response, which is designed to push the user back on course. However, if the data one is dealing with does have clarification material (*e.g.* a menu may also contain a little description of what a dish consists of as well as the name of that dish) one can catch these insertion sequences and offer the description as the system utterance. This has been done in the *Restaurant Take-away Menu Application*. Requests for meta-information are probably best dealt with through the help provided with the application.

4.6.1 □ Anaphoric references

The whole area of referring expression is of considerable interest, but practically, the only instance I needed to model in any detail is one anaphora. Many task-oriented dialogues are involved with the provision of information, and this is often in the form of lists. Humans have a strong tendency to use one anaphora to refer to this sort of data (*e.g.* ‘Give me the first one.’, ‘What about the second last one?’). By keeping an array of the order in which information is offered by the system, we should be able to model this phenomena. This remains future work.

Long distance pronominal anaphora, and other referring expressions in our types of dialogues are rare. Suitable grammar, and prompt writing can handle most of them.

4.6.11 Temporal Reference Resolution

In our domain, anaphoric temporal reference can be handled by suitable grammar, and prompt writing, but deictic references require a computational solution. Fortunately, this is quite within the powers of ECMAScript, although it does rely on the computer’s clock, so one can face time-zone issues. Deictic references cannot always be assumed to produce the same result in different contexts. In a flight context ‘Can I fly on Tuesday.’ would almost inevitably mean next Tuesday. In an interrogation context ‘When did you see John? – I saw him on Tuesday.’ would mean last Tuesday. It may be possible to gain clues from the tense of the verb, but in practice one can rely on a consistent result within any single domain and model accordingly. None of the dialogues I modeled presented this problem so that, while I have written ECMAScript capable of resolving deictic temporal references, it has not been employed.

4.6.12 Disfluencies

Many speech engines try to catch disfluencies (‘um’, ‘ah’, etc.), and discard them. This means that even if we wanted to try, and catch them, the data has ‘gone missing’. There is little choice but to fall back onto a style of dialogue that procures information item by item, if disfluencies cause a more general approach to fail.

4.6.13 Correction of □ rrors

The natural human strategy on error is to pause and correct, often without any clue that an error was present. □ajicek and Hewitt [1990] (not sighted) confirm that users prefer to repeat their input at least once before having to choose from a menu. Apparently, the

first reaction to an error should, therefore, be to allow the user a second chance. Thereafter, if the system is in a position to offer a short list of correct suggestions it should do so. None of the dialogues I implemented offered this opportunity. Spelling words is a normal human method of disambiguating badly heard words, and we have already dealt with it at *Section 4.5.2*. Finally, just as in human conversations (and computer applications), an undo-redo facility is really essential. I have modeled this in the *Restaurant Take-Away Menu Application* but a more general facility to say ‘check’ or the like at any point would be useful future work.

4.6.14 Confirmations

Confirmations play an important part in human dialogues but excessive confirmation creates boring exchanges. Some matters require legal confirmation, and, today, it is best to use touch-tone confirmation for these. In other cases, the introduction of the captured data item into the next prompt is quite natural: system - ‘What is your name?; user - ‘John.’; system - ‘OK, John □ .’. An appropriate response can be made if the grammar returns ‘wrong’ or ‘no’. We have already seen [Flammia 1998] that in some simple dialogues, nearly half of users dialogue turns were acknowledgments (*e.g.* “okay”, “alright”, “uh-huh”). In a perfect world, we would model these and somehow manipulate a confidence factor relating to what has so far taken place. However, this would require a large statistically based study of simple dialogues that is beyond the scope of this project. We have modeled a reasonable level of confirmation, and the opportunity to correct in the *Restaurant Take-Away Menu Application* – it does appear that different patterns will require different confirmation treatments.

4.6.15 Help

We have already dealt with the provision of help at *Section 4.4.4* above. A second aspect is the provision of tutorials. For different types of applications, and VoiceDBC itself is one, tutorials are essential to familiarise the user with the application. Speech applications that fall at the distant end of the dialogue-task distance spectrum are too simple to require tutorials.

4.□ Database Interfacing Issues

4.□.1 Introduction

Of course, VoiceDBC must simplify the problems of connecting to, analysing, and handling databases, and the data they contain. At one end of our applications, there are databases containing information in the form of strings, numbers, times, and so on. The strings will often be abbreviations, and they, together with most of the other data, have to be converted to a form suitable for ordinary text-to-speech processing. At the other end of our applications are natural words that require to be converted back into forms suitable for use in these databases. This gives us a framework upon which to base lexicons (which are used to handle the former task) and grammars (which are used to handle the latter task). So that VoiceDBC can be platform independent, I do not make use of any of the standard VoiceXML data types, all of which are (surprisingly) platform dependent. VoiceDBC contains graphical user interfaces that allow the user to contact the database in question, to analyse the data, characterise its data type and create lexicons. Currently the grammars are automatically constructed (using a terse approach – see *Section 4.7.2* below) from the lexical material.

4.2 The Terse Approach

Elsewhere in this thesis, we have explored the problems that expanding the vocabulary for an application causes in the form of increasing recognition errors. It is most unlikely, in the field of simple task-oriented dialogues, that we will have the resources to carry out a significant study of the vocabulary that is likely to be used. It is, therefore, best to adopt a policy of trying to get the user to use the words we want to hear, and these are the ones that arise from the nature of the data in the database, and the task at hand. This implies keeping the system utterances short, and to the point (terse), and politely (but firmly) reminding the user of what is allowed when they stray off course.

4.3 Lexicons

The approach used to create lexicons should be to review the data in each field in the database as it presents itself to the speech application. A currency value in Access may look like £18.00, but when recovered by a remote application presents itself as 18.0000. The destination city in a flight schedule presents itself as bri in the database, and has to be converted to Brisbane for the speech application. VoiceDBC provides a graphical user interfaces that allow a visual inspection to be made and each item to be lexicalised, that is mapped in a look up table (such as 'bri → Brisbane') or mapped by a Perl substitute function. It is useful if this transformation can be abstracted, and this can be done for some data types like currencies, times and dates. Where I came across these in the dialogues I implemented, I wrote functions to transform the data as it emerges from the database into strings suitable for text-to-speech modules.

4.4 Grammars

Grammars serve two purposes in VoiceXML. They specify what inputs the application will accept at each point in the dialogue, and in doing so they assist the speech recognising module to decide what it has heard. They also allow the programmer to allocate a return value, called the slot value, which can be different from the word(s) recognised. For example the grammar: '[no | nope | nine] |no| will always return 'no' as a slot value available for the application. If no slot value is specified the default is to the recognised word. The natural place to reverse lexicalisation is here, by using the database form as the slot value, and the natural language form as the grammar. While I would not wish to use the grammar shown above, as it would encourage a wider range of user utterances than desired, it does make a lot of sense to look at providing multiple introductory, and trailing words so that recognition does not simply fail as a result of someone saying "Can I fly?" rather than "Can I go?". Of course the full range of variations that are required can only come from a proper study of a relevant corpus so in the VoiceDBC case, we must rely upon the intuition of the designer.

4. Conclusions

In this chapter we have looked at the actual implementation of VoiceDBC. First, we looked at the general architecture (*Section 4.2*). Then we looked at implementing design patterns (*Section 4.3*). Next, we worked through the incorporation of good dialogue practice (*Section 4.4*). A section followed this on some of the general problems that the current state of speech recognition gives rise to (*Section 4.5*). Finally, we looked at dialogue phenomena (*Section 4.6*) and database interfacing (*Section 4.7*). During the chapter, I was careful to flag items for further work and these are fully listed at *Section 6.6* in the final chapter, *Conclusions*. In the next chapter we will look at VoiceDBC in action.

Chapter 5 - VoiceDBC in Action

5.1 Introduction

In the last chapter we looked at the implementation of VoiceDBC, but in this chapter we look at VoiceDBC in action. This will involve working through one application in detail from a user's point of view. It is rare to find simple solutions to complex problems, and VoiceDBC (like PowerBuilder, Visual Basic, Delphi, VBuilder, *et al.*) cannot be understood simply by opening it and clicking on a few buttons. It is essential that the user undertake the tutorials that accompany any of these applications to discover how they work. VoiceDBC's tutorial is reproduced as *Appendix II* and the reader is invited to at least skim it, to see in greater detail, how an application is produced.

The choice of application (a restaurant take-away menu) may seem strange but a *pizza ordering application* has become ubiquitous; I wanted to do something more challenging but along the same lines. An average Chinese restaurant menu is very much more complicated than a typical fast food menu and I thought it would be suitable.

5.2 The Restaurant Take-away Menu Application

As was seen in *Figure 11* at the start of the previous chapter, VoiceDBC opens with a dialogue box, which allows the user to choose to create a new speech application or open an existing one for further work. If the user chooses **ew**, VoiceDBC leads them through a wizard to capture the parameters essential to write the application. VoiceDBC comes with a tutorial (written in HTML) which can be accessed by clicking on **Help** **Tutorial**. The tutorial takes the user through the *Restaurant Take-away Menu Application*, which is one of the dialogue design patterns that is shipped with VoiceDBC.

5.2.1 The ature of the Restaurant Take-away Menu Application

It is useful to consider the nature of this dialogue before looking at how a user of VoiceDBC would use that tool to write it. Unfortunately, the standard VoiceXML form algorithm did not provide the sophistication to offer any real mixed initiative style of dialogue in this case so I split this function into two forms, the first handled the mixed initiative (*Figure 16*) and the second handled the offering of items dish by dish (*Figure 17*).

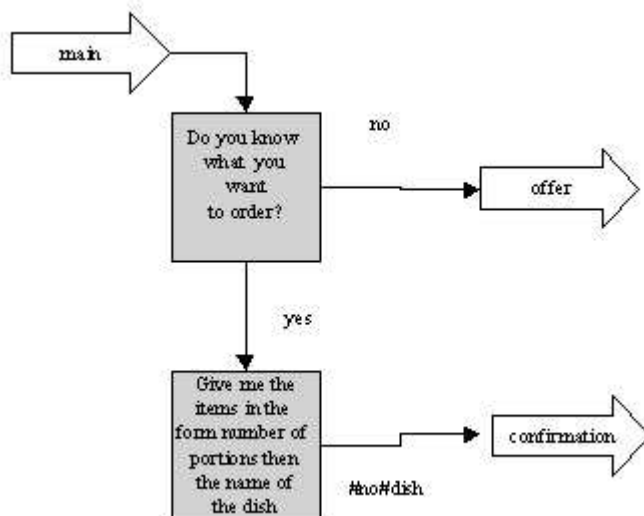


Figure 16 – The Mixed Initiative Form

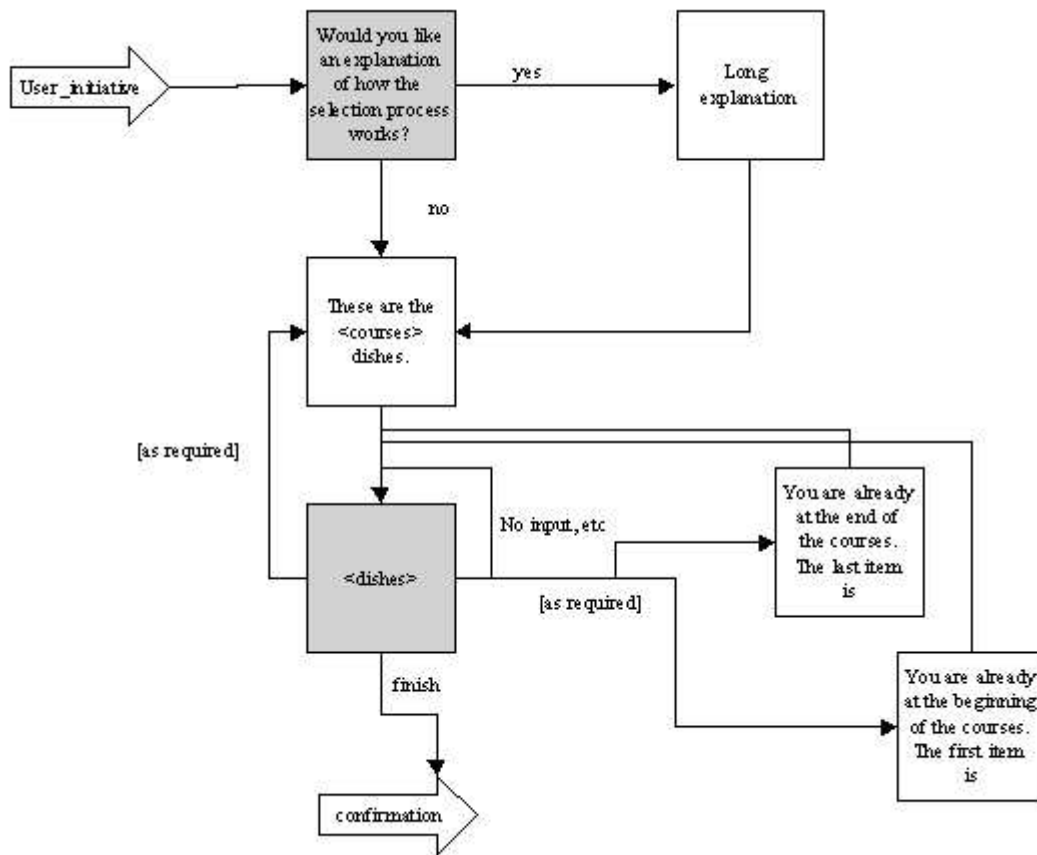


Figure 17 – The Offer Form

There was a third form (Figure 18) to handle the quite complicated matter of confirmation. As can be seen, the dialogue has been reduced largely to yes/no inputs from the user. This is because trying to write a grammar to accept all the variations on the typical contents of a (Chinese) restaurant is a nightmare. It is almost guaranteed not to work. Our expert can place an order by using the expressions used in the menu during the mixed initiative phase, but failing that, the only viable course is to reduce the exchange to yes/no (or numbers of portions). The dialogue produces quite different outcomes for the ‘expert’ or the ‘novice’ (defined here respectively as someone who knows how to use the menu item names and someone who does not).

Expert Exchange:

Computer: Do you know what you want to order?
 User: Yes.
 Computer: Give me the items in the form number of portions and then the name of the dish.
 User: One Spring Roll, One Prawn Toast, One Salt and Pepper Calamari Rings, One Peking Duck.
 Computer: Now, I will read back your order. One Spring Roll □ followed by a slight pause to allow the user to say ‘wrong’ □
 Computer: One Prawn Toast. □ followed by a slight pause to allow the user to say ‘wrong’ □

Computer: One Salt and Pepper Calamari Rings. □ followed by a slight pause to allow the user to say ‘wrong’□

Computer: One Peking Duck. □ followed by a slight pause to allow the user to say something□

Computer: OK, your order comes to □45.60. It will be ready for you to pick up in 30 minutes.

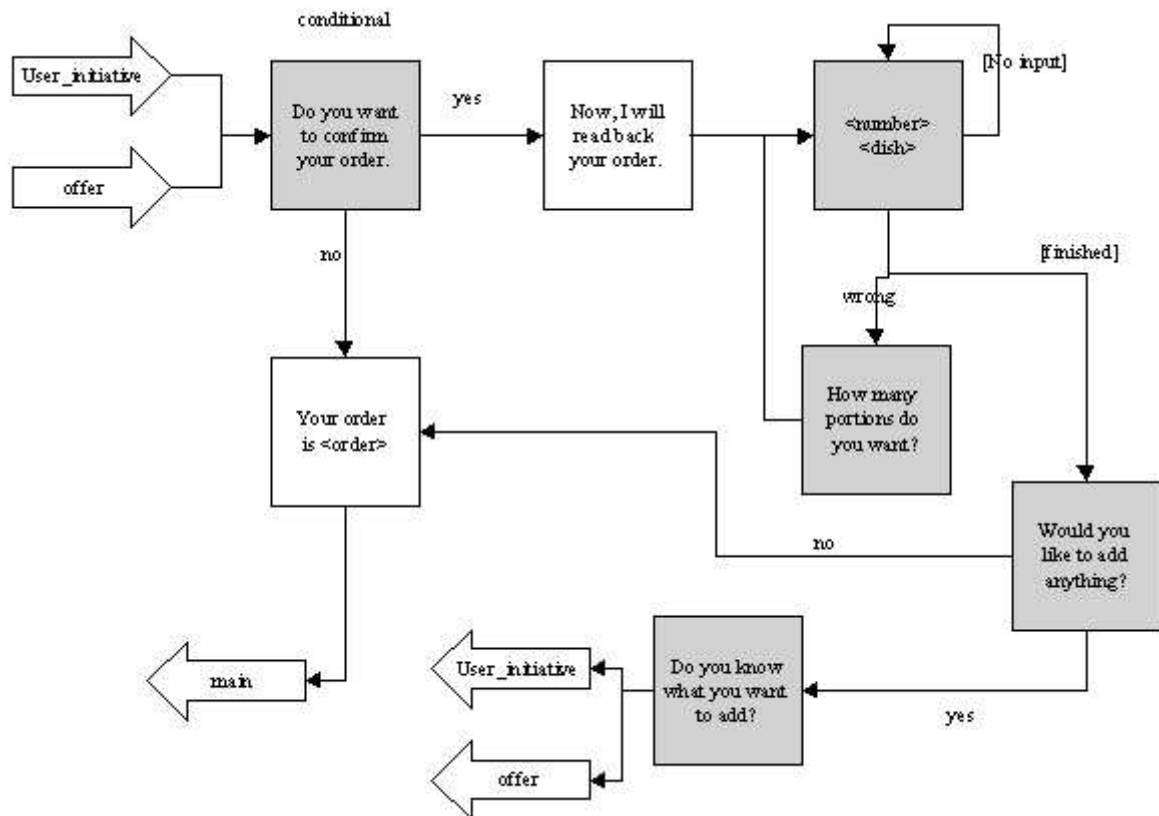


Figure 18 – The Confirmation Form

Novice Exchange

Computer: Do you know what you want to order?

User: No.

Computer: Would you like an explanation of how the ordering process works?

User: Yes.

Computer: I will go through the menu dish by dish. If you say nothing, no or next, I will go to the next dish. If you say any number, one through nine, I will record the number of portions of the dish you want. If you say yes, I assume you only want one portion. Dishes are arranged in courses, for example starters first, soups next. If you want to skip to the next course say ‘next course’. If you want to go backwards say ‘back up’. I will continue to go backwards until you say next. If you want to skip backwards a course say ‘last course’. Say finish to end and to have your selection read back for confirmation.

Computer: These are the starter dishes. San Chow Bow with Prawns and Scallops. □ followed by a slight pause to allow the user to say something□

Computer: San Chow Bow with Pork.

User: One.

Computer: Mixed Entree

User: One.
Computer: Spring Rolls.
User: Next course.
Computer: These are the soup dishes: Short soup.
User: Next course.
Computer: These are the Salt and Pepper dishes. Salt and Pepper Calamari Rings.
User: Next.
Computer: Salt and Pepper Tofu.
User: Next.
Computer: Salt and Pepper Pork Chops.
User: One.
Computer: Salt and Pepper Prawns
□ followed by a slight pause to allow the user to say something□
Computer: Salt and Pepper Scallops.
□ followed by a slight pause to allow the user to say something□
Computer: These are the Duck Dishes. Peking Duck.
User: One.
Computer: Fried Duck with Plum Sauce.
User: Finish.
Computer: Now I will read back your order. Say wrong if you disagree. One San
Chow Bow with Pork.
□ followed by a slight pause to allow the user to say something□
Computer: One Mixed Entree.
□ followed by a slight pause to allow the user to say something□
Computer: One Salt and Pepper Pork Chops.
□ followed by a slight pause to allow the user to say something□
Computer: One Peking Duck.
User: Wrong.
Computer: How many portions do you want?
User: Two.
Computer: Would you like to add anything?
User: Yes.
Computer: Do you know what you want to add?
User: Yes.
Computer: Give me the items in the form number of portions and then the name of
the dish.
User: One Fried Duck with Plum Sauce.
Computer: Do you want to reconfirm your order?
User: No.
Computer: OK, your order comes to □127.40. It will be ready for you to pick up in
30 minutes.

As can be seen, the implementation can give rise to widely different dialogues.

5.2.2 Using VoiceDBC to Write the Dialogue

Appendix II contains the tutorial that is shipped with VoiceDBC and which covers the writing of the Restaurant Take-away Menu Application, so we will not go through every step here, but will concentrate on some of the more interesting aspects. The first three forms, (I use the word *form* to refer to a single graphical user interface within the series of these which constitute the wizard) of the wizard allow one to name the project, to set up any opening and closing remarks, to choose the basic dialogue design pattern from a

template browser, and to specify paths to the cgi-bin, and the database that will be used. VoiceDBC then contacts the database, obtaining its contents and displays the form at *Figure 19*.

New Project Wizard

Review the Data Generally

VoiceDBC is displaying the fields which make up the table you specified. You have to choose datatypes for them. Click on Analyse to do this. The label will change to Visited when you return to this form.

First you must provide natural language expressions for each field. The database field names are shown in the first column and the default NL references in the data entry boxes in the second column. Overtyping any you wish to change.

primary_key	primary_key	Add Words	Analyse
courses	courses	Add Words	Analyse
dishes	dishes	Add Words	Analyse
descriptions	descriptions	Add Words	Analyse
price	price	Add Words	Analyse

Help < Back Next > Cancel

Figure 19 – Review the Data Generally Form from the New Project Wizard

This shows the name by which the field is referred to in the database, and offers the name back as a default lexicalisation. We can overtype the default lexicalisation and VoiceDBC will use the new description in the application. We then analyse each field in turn. We will only look at *courses* and *price*, and see how to lexicalise or transform a string or characterise a field as a currency, for future use in the application. If we click on Analyse on the course line, the form shown at *Figure 20* is displayed. This allows us to view the data, (and in this case it is in a fairly natural language form). If it were not for the fact that we wish to see how the lexicalisation, and transformation screens work, we might just select String, and Map Directly, but we will select String, and Lexicalise instead, and click on Next.

As can be see, at *Figure 21*, the new screen that is displayed affords us the opportunity to enter a lexical reference for each unique string item in the database. Of course, in this case nothing needs to be done but, whatever is left appearing in the white box when we leave this form, is remembered for future use when converting database tokens into specifications of speech output by VoiceDBC.

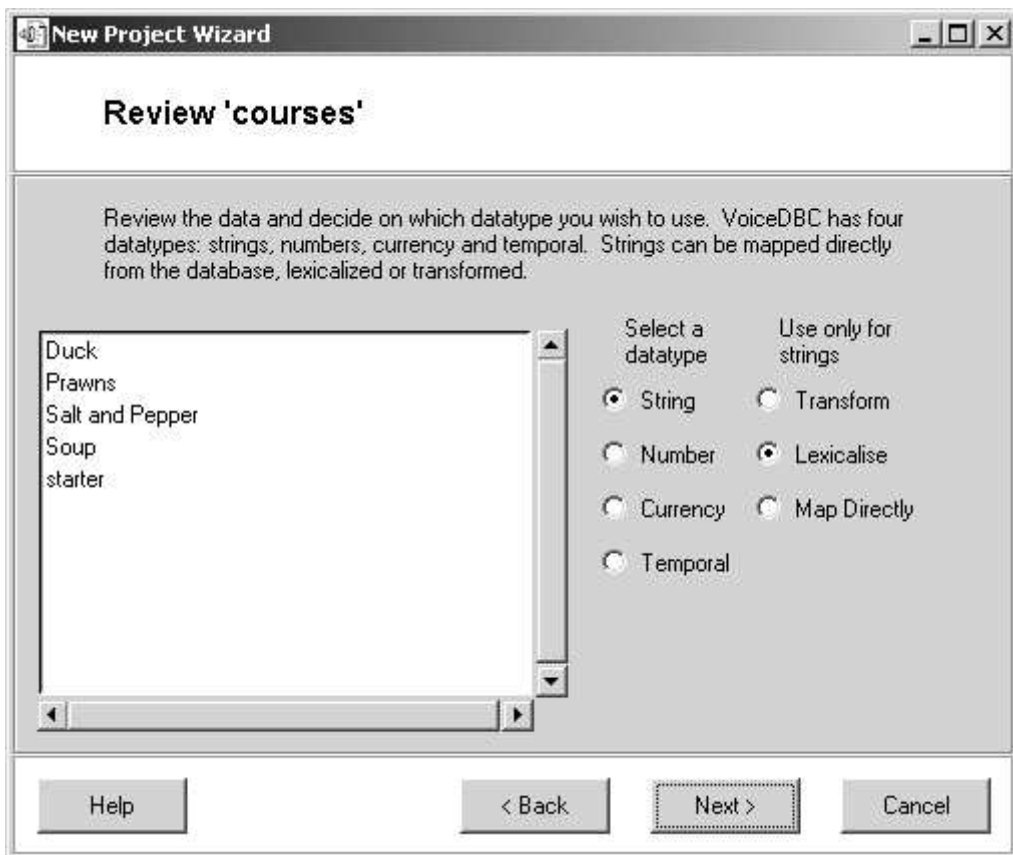
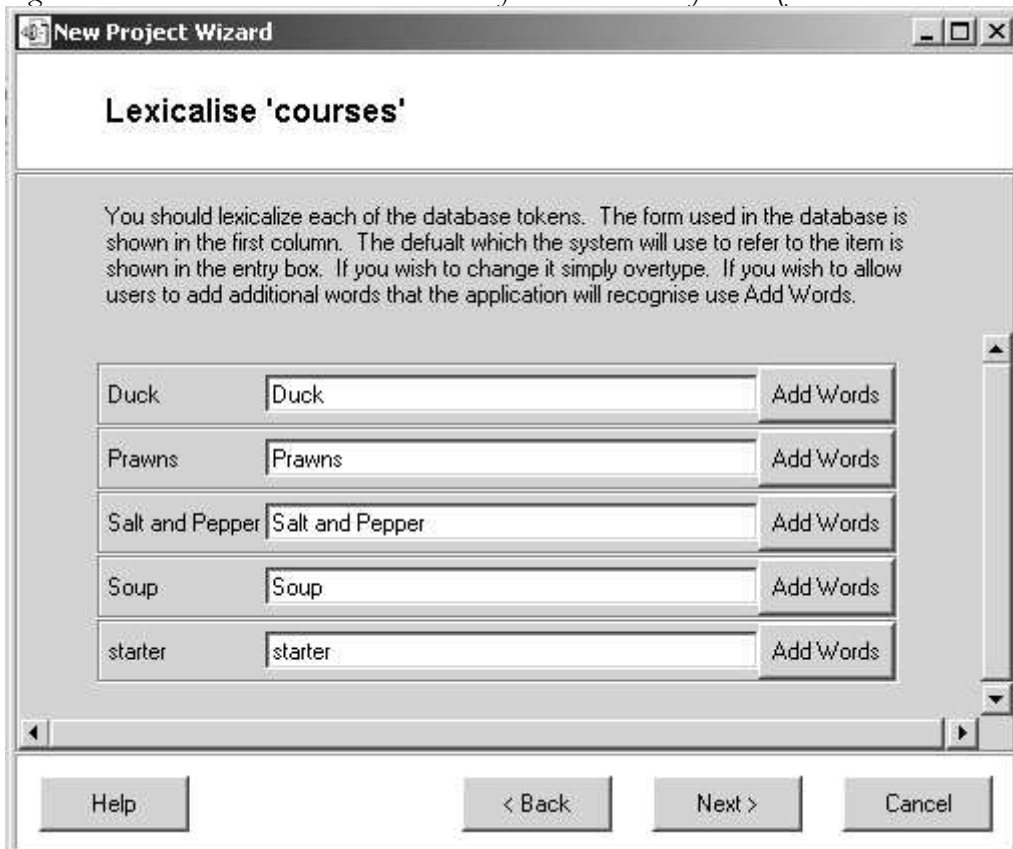


Figure 20 – The Review ‘courses’ Form from the New Project Wizard – above

Figure 21 – The Lexicalise ‘courses’ Form from the New Project Wizard - below



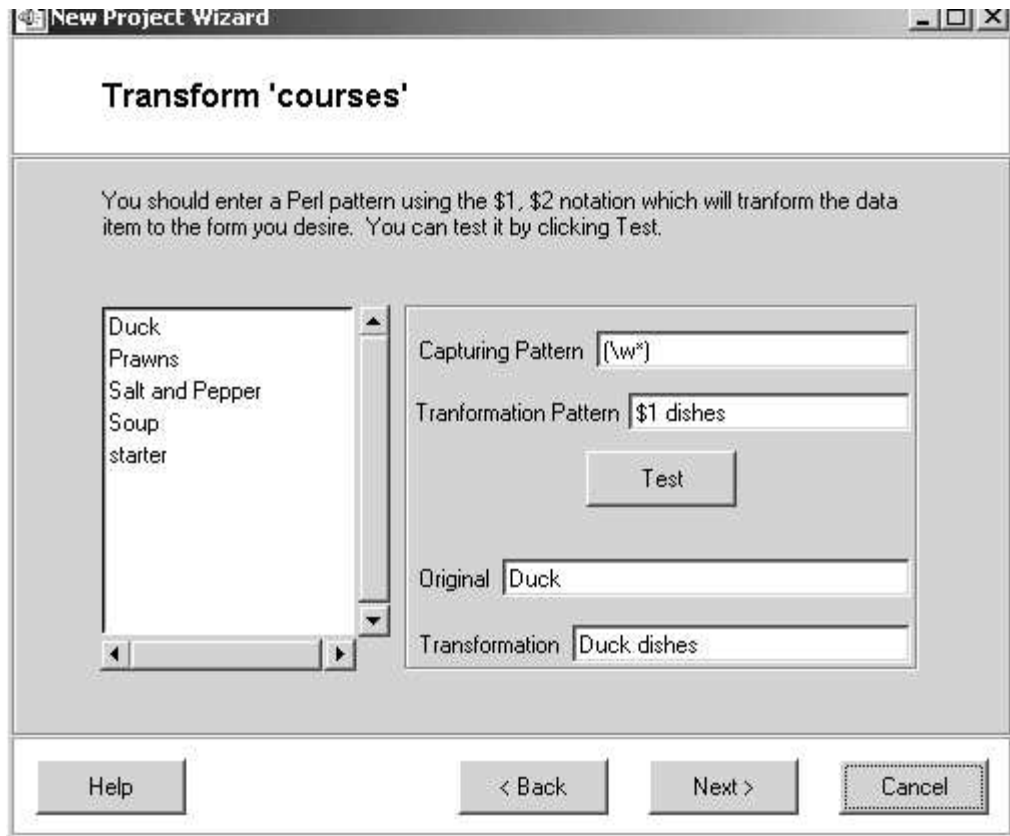
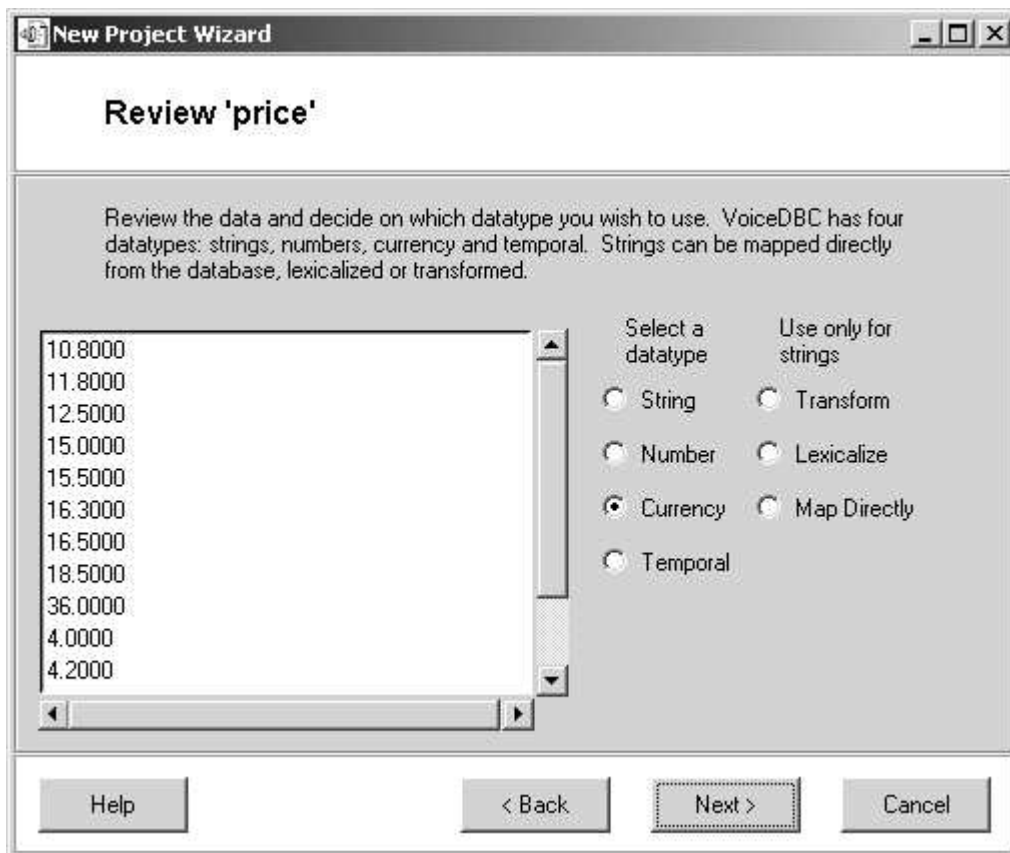


Figure 22 – The 'Transform courses' Form from the New Project Wizard – above
 Figure 23 – The Review 'price' Form from the New Project Wizard



If we want to handle this mapping from the database to the speech application differently, we can use transformation. To add the word ‘dishes’ to every course prior to sending the data off, we select Transform on the Review form, and the screen shown at *Figure 22* is displayed.

As can be seen, I have entered a Perl pattern in the box titled ‘Capture Pattern’, and a further one in the box titled ‘Transformation Pattern’. This data is then used in a Perl substitute function that can be tested by clicking on Test. If we leave this as the transformation, any data item that appears in this field in the database will always have dishes added after it as it passes through the part of the code that handles the mappings.

The last data handling we look at is the currency one. The screen is shown at *Figure 23*, and it can be seen that the first item in the list shows as 10.5000. We know it is a currency item, and this view reveals that inside Access numbers are stored as floating point numbers (to four decimal places). If we select Currency, the figure will be transformed to \$10.50 upon transfer to our speech application, and can be rendered to speech by a text-to-speech module, normally as ‘ten dollars fifty cents’.

The only thing that is left to do is to decide which parts of the database should be offered to the user, what descriptions should be provided to any clarification question, and how grouping should take place. If we click Next twice we see *Figure 24*.

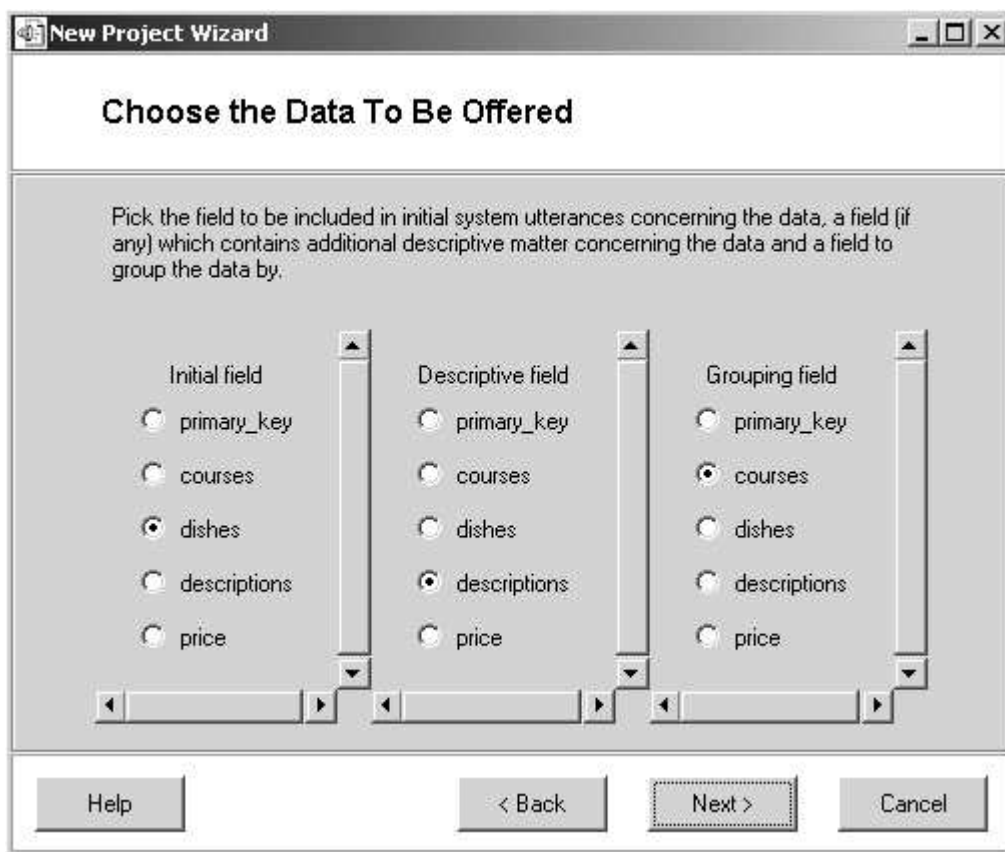


Figure 24 – The Choose the Data To Be Offered Form from the New Project Wizard

The choices in this case are quite logical, and, with two more clicks on Next, VoiceDBC write all the code required for the application. The documents are offered to the developer as is shown in *Figure 25*. This task has been completed within minutes and it may well take longer to deploy the documents to the VoiceXML Gateway and the cgi-bin

(and change the permissions on the files) than actually write the application is the first place.

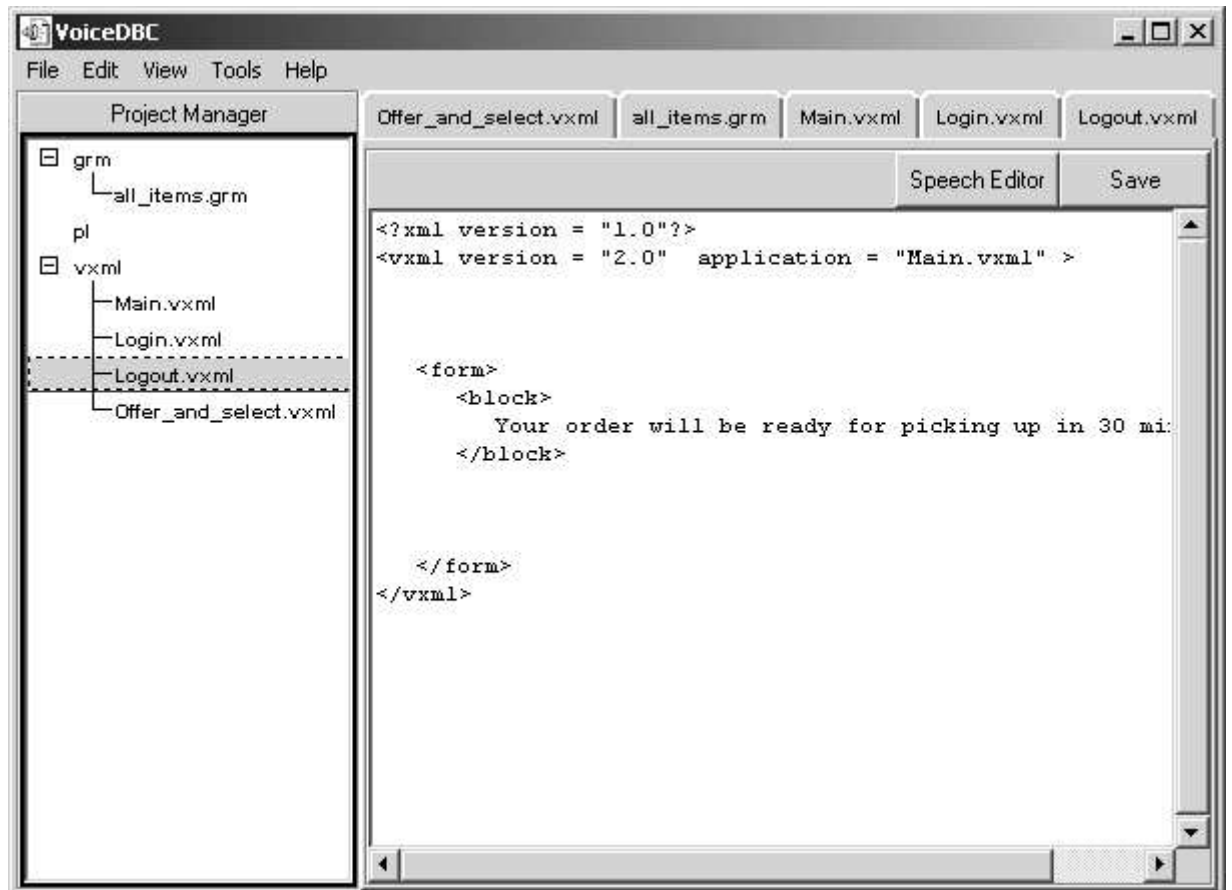


Figure 25 – The final documents being displayed by VoiceDBC.

5.3 The Catalogue Sales Application

The Restaurant Take-away Menu Application was used as the basis for a catalogue sales application. This uses a database that contains information on electrical goods. The finished application can be found on the accompanying CD. With changes mainly from *dish* to *item*, and *course* to *groups*, the templates made the transitions with little problems. The application can be reviewed by clicking **File** **Open** **Existing Project**, and then selecting **catalogue sales** and clicking **Open**. The individual text files can be opened simply by double clicking on the file name in the **Project Manager**.

5.4 Conclusions

In this chapter we have looked at VoiceDBC and at one particular application, *The Restaurant Take-away Menu Application*, in some considerable detail. Further detail is contained in *Appendix II*, which contains the full tutorial that is shipped with VoiceDBC. We then briefly touched upon *The Catalogue Sales Application*, which can be reviewed more fully off the CD. In the next chapter, we will conclude this thesis.

Chapter 6 – Conclusions

6.1 Outcomes

When I started this project some eight months ago, I specified in the project proposal that the aims were:

“This project aims to remedy the deficiency in current VoiceXML tools by:

- Collecting and collating a specification for good dialogue practice.
- Identifying relevant characteristic dialogue phenomena and producing algorithms and strategies to handle them.

Using this knowledge it then aims to specify, design and implement an integrated development environment – called VoiceDBC - which will also support easy database connectivity.”

These objectives have been achieved. As *Chapter 2* shows, I have collected, and collated a large body of literature, and applications relevant to the subject. I did not go onto write a formal specification for good dialogue practice; with work around like Balentine and Morgan [1999], to do so would have been prolix. VoiceDBC has been produced, and writes speech applications. In addition it handles database connectivity. The outcomes of research are always (indeed should always be) unpredictable. The work on dialogue design patterns was an unexpected, and interesting bonus to this work.

Of course, there is still much that can be achieved with VoiceDBC. In addition to the usual developer’s to-do list, mainly involved with house-keeping matters, like making sure that buttons are disabled when they should be and that boxes that should not accept text do not, some ten significant items on completing the work on incorporating good dialogue practice, and dialogue phenomena are listed in *Section 6.6* below. In addition, a number of other areas, some directly concerned with VoiceDBC, like completing the work on templates, and some concerned with speech application tools that might come after, like a canvas based tool for task-oriented dialogues with a short dialogue-task distance, are also considered below.

6.2 Dialogue Design Patterns

The idea of dialogue design patterns was one of the most interesting ones that emerged from this project. I choose to use the bottom up approach to learn more about them. From that experience it is possible to conclude that many dialogue designs patterns are re-usable. Indeed, some components of task-oriented dialogues with a long task-dialogue distance turned out to be so standard, such as a greeting and a farewell, they could be re-use across all our dialogues. Of course, the approach taken can give rise to maintenance issues. Amendments in one template may have to be reflected in others, and if one had 30 or 40 templates, this could become a large task. Given the potential for this approach to simplify the writing of dialogues, it is clearly worthwhile to undertake the top down work considered at *Section 3.2.3*, and produce a more robust approach to catching and implementing dialogue design patterns.

6.3 A Visual Front End for VoiceDBC

I was intrigued by the idea of a highly visual front end to VoiceDBC that would allow the developer to work at a high level of abstraction. The point here does not involve good dialogue practice, but has to do with the way these tools allow us to think at a higher level, and manage the important aspects of the dialogue - the states and transitions - without getting bogged down in code. The work done by Kolzer [1999] gives a clear direction forward, and to bring to that, what has been learned about good dialogue practice, and to develop techniques to support compliance procedures, would constitute a fascinating piece of work. Perl modules do contain graphical user interface objects such as ‘canvas’, which appear quite capable of handling this type of interface. Such an interface is unnecessary given the use of dialogue design patterns, but would be interesting to enable work on dialogues with a shorter task-dialogue distance.

6.4 Expanding the Library of Templates

The taxonomy contained some of the many dialogue design patterns that might be worth capturing. This is an area that will empower VoiceDBC and allow it to satisfy a growing demand for good cheap speech application. Even if the top down work is never done it will still be worthwhile implementing simpler dialogues using VoiceDBC so that the work involved in capturing their patterns can become available for future use.

6.5 Implementing Database Connections to Other DBMSs

VoiceDBC currently only has connectivity with the Microsoft’s Access Database. However, it uses the Perl DBI modules that are designed for easy access with all the major databases. It would be most useful to implement connectivity with mySQL (which is proving a very popular free DBMS for both Windows and Linux platforms) and for ORACLE with its massive market presence.

6.6 Completing the Work on Incorporating Good Dialogue Practice and Dialogue Phenomena

There were ten items flagged under these headings in *Chapter 4*. These are:

- Production of a checklist for compliance with good dialogue practice suitable for use when writing task-oriented dialogues with a long dialogue-task distance. – *Section 4.4.1*
- Completion of the work on tapering prompts and the various other system responses based upon the number of times they are repeated. – *Section 4.4.4*
- Production of a style guide. – *Section 4.4.7*
- Input by spelling (including a fall back to the keypad). – *Section 4.5.2*
- Discriminating between expert–novice users involving the maintenance of an application specific database. – *Section 4.6.2*
- Incorporating a general repeat function – *Section 4.6.3*
- Incorporation of a form writing module into VoiceDBC. – *Section 4.6.3*
- Inclusion of a sleep mode, and a slower rate of speaking. – *Section 4.6.4*
- Creation of a one-anaphora management module. – *Section 4.6.10*
- A general application wide undo–redo facility. – *Section 4.6.13*

6.□ Conclusions

One day the speech recognition problem will be solved, and, when that happens, the problems of understanding language will become the main focus when creating speech applications. In the meantime, it is much more important to build robust speech applications that deliver what they can truly promise. Tools such as VoiceDBC, which can semi-automatically produce speech applications that are robust, will play their part in ensuring that the promise of this field is realised.

It is true that the range of speech applications that VoiceDBC can produce is currently limited. However, if the work is done to expand the range of templates, VoiceDBC can cover most of the field from travel booking to pizza ordering, taxi booking to sales fulfillment, news, weather, and timetables information to immigration facts. Moreover, it will allow speech applications that conform to good dialogue practice to be built quickly by anyone willing to follow the tutorial.

References

- Abbott K R. (2002). *Voice Enabling Web Application* □ *VoiceXML and Beyond* pages 87 to 103. Springer-Verlag, New York.
- Alexander C. (1977). *A pattern language* □ *Towns, Buildings, Constructions*. Oxford University Press.
- Alexander C. (1979). *The Timeless Way*. Oxford University Press.
- Andersson E A, Breitenbach S, Burd T, Chidambaram N, Houle P, Newsome D, Tang X and □hu X. (2001). *Early Adopter VoiceXML*, pages 113 to 133. Wrox Press, Birmingham, England.
- Androutopoulos I, Ritchie G D, Thanisch P. (1994). *Natural Language Interfaces to Databases – An Introduction*, Research Paper 709. Department of Artificial Intelligence, University of Edinburgh, Scotland.
- Balentine B and Morgan D P. (1999). *□ow to Build a Speech Recognition Application*. Enterprise Integration Group, San Ramon, California.
- Barto A G, Bradtke S J and Singh S P. (1995). *Learning to act using real-time dynamic programming*. Artificial Intelligence Journal, 72 (1-2), pages 81 – 138.
- Beasley R, Farley M, O'Reilly J and Squire L. (2002). *Voice Application Development with VoiceXML*, pages 49 to 110. Sams Publishing, Indianapolis.
- Bellman R E. (1957). *Dynamic Programming*. Princeton University Press, Princeton, N.J.
- Bernsen N O, Dybkjaer H, Dybkjaer L. (1998). *Designing Interactive Speech Applications – From First Ideas to User Testing*. Springer Verlag.
- Buskirk R V and LaLomia M. (1995). The Just Noticeable Difference of Speech Recognition Accuracy. in *proceedings of ACM □ □ I95* (Poster), page 95.
- Dahlback N. (1997). *Towards A Dialogue Taxonomy*. Department of Computing and Information Science, Linkoping University, Sweden
- Dale R and Reiter E. (1996). The Role of the Gricean Maxims in the Generation of Referring Expressions. Pages 16-20 in *Working Notes for the AAAI Spring Symposium on Computational Implicature* □ *Computational Approaches to Interpreting and Generating Conversational Implicature*, Stanford, March 25-27.
- DISC. (1999,2000). <http://www.disc2.dk> accessed various times during 2002.
- Edgar B. (2001). *The VoiceXML □ andbook – Understanding and the Building Phone-Enabled Web*. CMP Books, New York.
- ETSI DES □HF-00021 □ 0.0.40. (2002). *□uman Factors* □ *Under Interfaces* □ *Generic spoken command vocabulary for ICT devices and services*. <http://portal.etsi.org □HF □STFs □STF182.asp> accessed in May 2002.
- Eisenzopf J. (2002). Top 10 Best Practices for Voice User Interface Design. in *VoiceXMLPlanet*, □une 2002, <http://www.voicexmlplanet.com □articles □bestpractices.html>.
- Flammia G. (1998). *Discourse Segmentation of Spoken Dialogue* □ *An Empirical Approach*. Ph.D Thesis, MIT.

- Frankish C, Hull R, and Morgan P. (1995). Recognition Accuracy and User Acceptance of Pen Interfaces. In *Proceedings of ACM CHI 95* pages 503 – 510.
- Fraser N. (1997). Assessment of interactive systems, pages 564 to 615 in Gibbon D, Moore R and Winski R, (ed) *Handbook of Standards and Resources for Spoken Language Systems*. Mouton de Gruyter, New York.
- Gamma E, Helm R, Vlissides J and Jones R. (1994). *Design Patterns*. Addison-Wesley, N.Y.
- Glass J R. (1999). *Challenges for Spoken Dialogue Systems*. Spoken Language Systems Group, MIT Laboratory for Computer Science, <http://www.sls.lcs.mit.edu>, accessed May 2002.
- Grand M. (1998). *Pattern in Java – Volume 1 – A catalogue of reusable Design Patterns Illustrated with UML*. John Wiley & Sons, Inc, N.Y.
- Grice H P. (1975). Logic and Conversations, In Cole P and Morgan J, (ed), *Syntax and Semantics: Vol .3, Speech Acts*, pages 43 – 58. Academic Press, New York.
- Guelich S, Gundavavaram S and Birznieks G. (2000). *CGI Programming with Perl*. O'Reilly, CA.
- Hild H and Waibel A. (1996). *Recognition of Spelled Names Over The Telephone*. Interactive Systems Laboratories, University of Karlsruhe, Germany and Carnegie Mellon University, Pittsburg, U.S.A..
- Hobbs JR. (1978). *Resolving Pronoun References*. *Lingua*, 44, 311-338.
- Hood D. (2001). Spoken Dialog Systems.
<http://www.ics.mq.edu.au/~dhood.systems.html>. Accessed 27 October 2002
- Hulstijn J. (2000). *Modeling Usability – Development Methods for Dialogue Systems*. Computer Science, University of Twente.
- Kennedy C and Bogoraev B. (1996a). Anaphora in a wider context: Tracking discourse referents. In *Proceedings of the 12th European Conference on Artificial Intelligence*, pp582-586, August 11-16, Hungary.
- Kennedy C and Bogoraev B. (1996b). Anaphora for everyone: Pronominal anaphora resolution without a parser. In *Proceedings of the 16th International Conference on Computational Linguistics*, Copenhagen, Denmark.
- Keeney R and Raiffa H. (1976). *Decisions with Multiple Objectives – Preferences and Value Tradeoffs*. John Wiley and Sons.
- Kolzer A. (1999). Universal Dialogue Specification for Conversational Systems. *Linköping Electronic Articles in Computer and Information Science, Vol 4 (1999) nr 28*.
<http://www.ep.liu.se/~ea/cis/1999/028> December 30, 1999.
- Lappin S and Leass H J. (1994). An Algorithm for Pronominal Anaphora Resolution. *Computational Linguistics*, Volume 20, Part 4, pages 535-561.
- Mankoff J and Abowd G D. (1999). *Error Correction Techniques for Handwriting, Speech and other ambiguous or error prone systems*. GVU Center and College of Computing, Georgia Institute of Technology,
<http://www.cc.gatech.edu/fce/publications/interact99/index.html>.

- McKinlay A, Arnott J, Procter R, Masting O and Woodburn R. (1993). *A Study Of Turn-Taking In A Computer-Supported Group Task*. Dept. of Maths. and Computer Science, Dundee University, Scotland.
- Mittendorfer M, Niklfeld G, Winiwarter W. (2001). *Evaluation of Intelligent Component Technologies for VoiceXML Application.*, Software Competence Centre, Hagenberg.
- Norton L M, Weir C E, Scholz K W, Dahl D A and Bouzid A. (1996). A methodology for Application Development for Spoken Language Systems. In *Proceedings, Fourth International Conference on Spoken Language Processing*.
- Ohrstrom-Sandgren T, Weibe J, O'Hara T and McKeever K. (1997). *Temporal Resolution Algorithm*. Department of Computer Science and the Computing Research Laboratory, New Mexico State University, U.S.A.
- Ramakrishnan N, Capra R and Perez-Quinones. (2001). *Mixed-Initiative Interaction □ Mixed Computation*. Department of Computer Science, Virginia tech, Blacksburg, Virginia, U.S.A.
- Russell S and Norvig P. (1989). *Artificial Intelligence □ A Modern Approach*. Prentice Hall, N.J.
- Sharma C and Kunins J. (2002). *VoiceXML □ Strategies and Techni□ues for Effective Voice Application Development with VoiceXML 2.0*. Pages 329 to 373, John Wiley □ Sons, Inc., New York.
- Somerville I. (1989). *Software Engineering.*, Addison-Wesley Publishing Co, N.Y.
- Stone M. (2000). *Towards A Computational Account of □nowledge, Action and Inference in Instructions*. Department of Computer Science and Center for Cognitive Science, Rutgers, State University of New Jersey, U.S.A.
- Sutton R S. (1991). Planning by incremental dynamic programming. In *Proc. Ninth Conf. On Machine learning*, pages 353 – 357, Morgan-Kaufmann.
- TRINDI. (2001). <http://www.ling.su.se/projekt/trindi>, accessed May 2002.
- Rosenfeld R, Olsen D and Rudnicky A. (2001). Universal Speech Interfaces, in *Interactions, November □ December 2001*, <http://icie.cs.byu.edu/Papers/UniversalSpeech.pdf> accessed May 2002.
- Walker M A, Fromer J C and Narayanan S. (1998). Learning Optimal Dialogue Strategies: A Case Study of a Spoken Dialogue Agent for Email. In *proceedings of ACL □ COLING98*.
- Yankelovich N, Levow G and Marx M. (1995). Designing SpeechActs: Issues in Speech User Interfaces. In *Proceedings of C □ I95 Conference on □uman Factors in Computing Systems*, Denver.
- Weibe J, Farwell D, Villa D, Chen J L, Sinclair R, Ohrstrom-Sandgred T, Stein G, □arazua D and O'Hara T. (1996). ARTWORK: Discourse processing in machine translation of dialog. *Final technical report □3rd year □ MCCS-96-294* (Computing Research Lab), NMSU.
- Weibe J, Farwell D, O'Hara T, McKeever K and Ohrstrom-Sandgred T. (1997). An Empirical Approach to Temporal Reference Resolution. In *Proceedings Second Conference on Empirical Methods in Natural Language Processing*.

- Weibe J M, O'hara T P, Ohrstrom-Sandgren T and McKeever K J. (1998). An Empirical Approach to Temporal Reference Resolution. In *Journal of Artificial Intelligence Research* (9), pages 247-293.
- ajicek M and Hewitt J. (1990). An Investigation into the use of Error recovery Dialogues in a User Interface Management System for Speech Recognition. In *Proceedings of IFIP INTERACT90* pages 755-760.
- oltan-Ford E. (1991). How to Get People to Say and Type What Computers Can Understand. *International Journal of Man-Machine Studies* 34(4), pages 527-547.

Appendix I

CODIAL: the guidelines for cooperative dialogue

<p>Guidelines for cooperative dialogue</p> <p>The two tables below show the 13 generic guidelines (GG) and 11 specific guidelines (SG) for cooperative human-machine dialogue interaction. The generic guidelines are expressed at the same level of generality as are the Gricean maxims (marked with an □) [Grice 1975]. Each specific guideline is subsumed by a generic guideline. The left-hand column characterises the aspect of interaction addressed by each guideline.</p> <p>The guidelines in the tables represent a first approximation to an operational definition of system cooperativity in task-oriented, shared-goal interaction. Their purpose is that of achieving the shared goal as directly and smoothly as possible. It is exactly when a guideline is violated that miscommunication is likely to occur, which again may seriously damage the user's task performance. Roughly speaking the generic guidelines express what to take into account while the specific guidelines explain how to do it. Since all specific guidelines are subsumed by generic guidelines the user might decide only to use the generic guidelines. This would mean a more coarse-grained and overall characterisation of cooperativity issues but it would also mean fewer guidelines to remember and to apply.</p>		
Aspect	Generic Guideline	Paraphrase
Informativeness (aspect 1)	GG1	□Make your contribution as informative as is required (for the current purposes of the exchange).
	GG2	□Do not make your contribution more informative than is required.
Truth and evidence (aspect 2)	GG3	□Do not say what you believe to be false.
	GG4	□Do not say that for which you lack adequate evidence.
Relevance (aspect 3)	GG5	□Be relevant, i.e. be appropriate to the immediate needs at each stage of the transaction.
Manner (aspect 4)	GG6	□Avoid obscurity of expression.
	GG7	□Avoid ambiguity.
	GG8	□Be brief (avoid unnecessary prolixity).
	GG9	□Be orderly.
Partner asymmetry (aspect 5)	GG10	Inform the users of important non-normal characteristics which they should take into account in order to behave cooperatively in spoken interaction. Ensure the feasibility of what is required of them.
Background Knowledge (aspect 6)	GG11	Take partners' relevant background knowledge into account.
	GG12	Take into account legitimate partner expectations as to your own background knowledge.
Meta-communication (aspect 7)	GG13	Enable repair or clarification meta-communication in case of communication failure.

Aspect	Specific Guideline	Paraphrase
Informativeness (aspect 1)	SG1 (GG1)	Be fully explicit in communicating to users the commitments they have made.
	SG2 (GG1)	Provide feedback on each piece of information provided by the user.
Manner (aspect 4)	SG3 (GG7)	Provide same formulation of the same question (or address) to users everywhere in the system's interaction turns.
Partner asymmetry (aspect 5)	SG4 (GG10)	Provide clear and comprehensible communication of what the system can and cannot do.
	SG5 (GG10)	Provide clear and sufficient instructions to users on how to interact with the system.
Background Knowledge (aspect 6)	SG6 (GG11)	Take into account possible (and possibly erroneous) user inferences by analogy from related task domains.
	SG7 (GG11)	Separate whenever possible between the needs of novice and expert users (user-adaptive interaction).
	SG8 (GG12)	Provide sufficient task domain knowledge and inference.
Meta-communication (aspect 7)	SG9 (GG13)	Initiate repair meta-communication if system understanding has failed.
	SG10 (GG13)	Initiate clarification meta-communication in case of inconsistent user input.
	SG11 (GG13)	Initiate clarification meta-communication in case of ambiguous user input.

Appendix II – The VoiceDBC Tutorial

VoiceDBC's Tutorial

Welcome to VoiceDBC. It is rare to find simple solutions to complex problems and VoiceDBC (like other powerful development environments) cannot be understood simply by opening it and clicking on a few buttons. It is essential that the user undertakes the tutorials that accompany any of these application to discover how they work.

The Restaurant Take-Away Menu Problem

Producing a speech application which can successfully offer the contents of standard chinese restaurant's menu is quite challenging. This tutorial will take you through the production of such an application. In order to carry out the tutorial you must have Microsoft Access on your computer and you must use the database restaurant.wdb which is shipped with VoiceDBC in the directory *sample\databases*

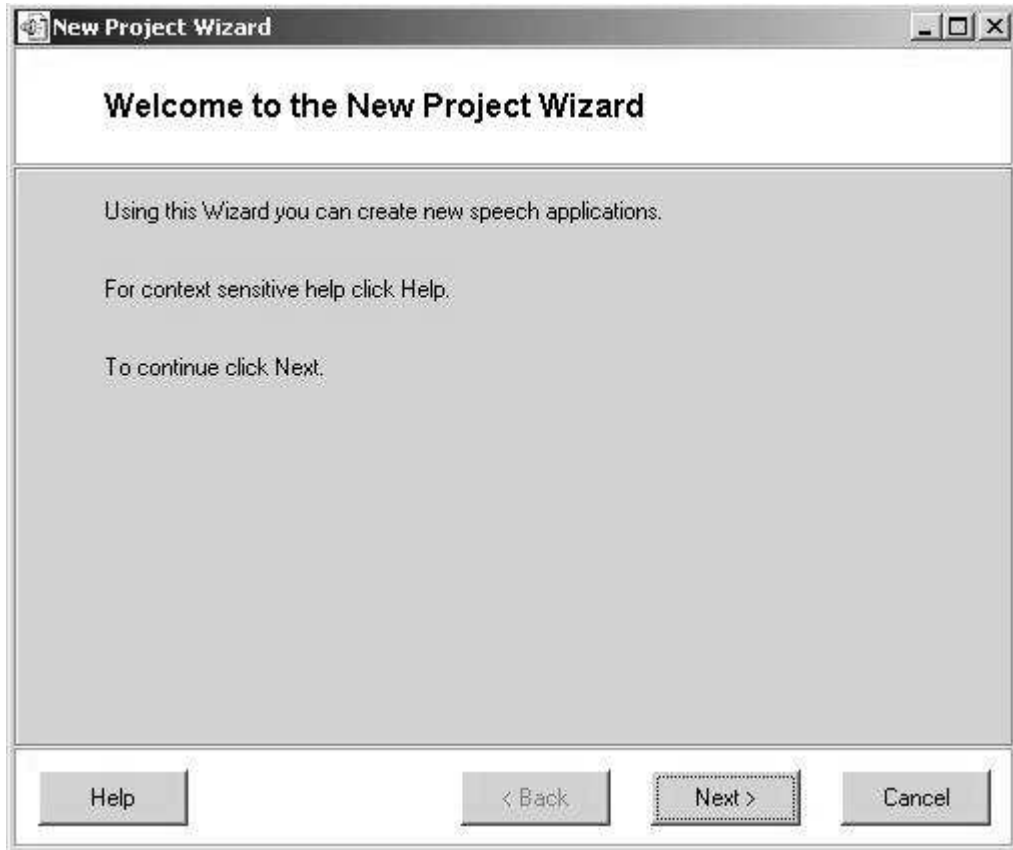
Getting Started

Open VoiceDBC by clicking on its icon or typing `perl VoiceDBC.pl` at the command line in the directory VoiceDBC. You should see the following screen:

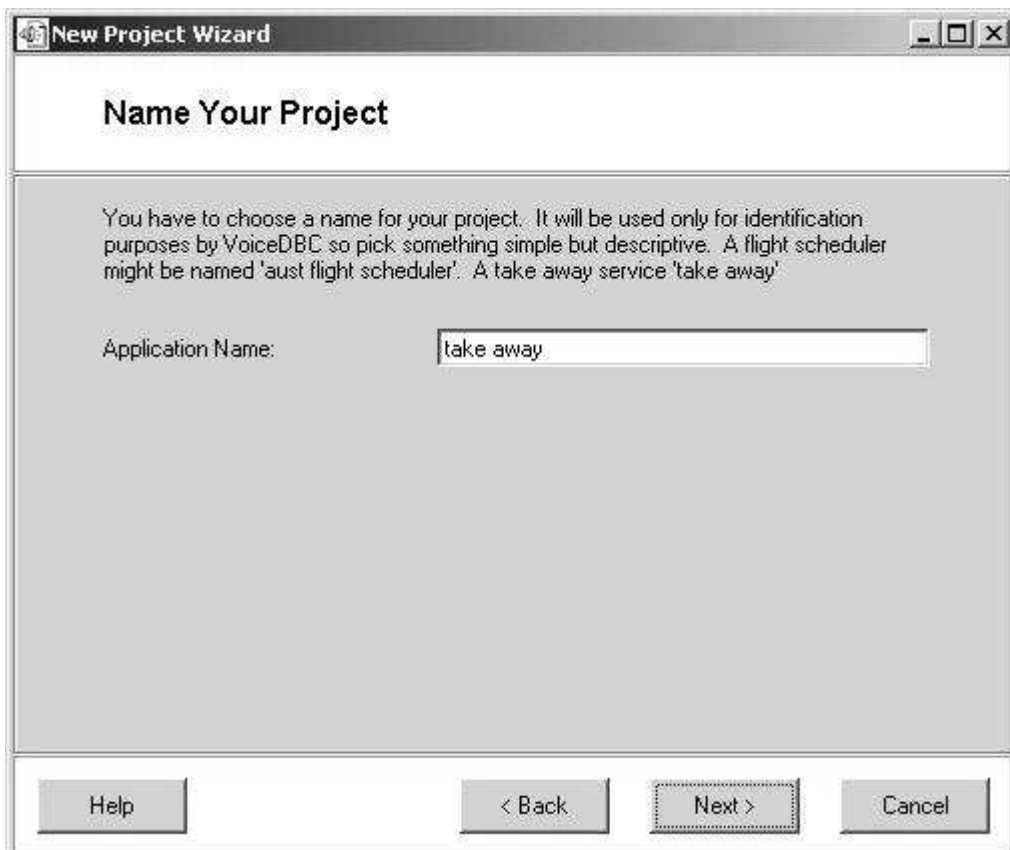


Click on **New**

Specifying Information



Click on **Next**. Although VoiceDBC will take care of the general framework of your application you must enter key utterances and parameters. First you must enter a name for your project.



Although VoiceDBC will take care of the general framework of your application you must enter key utterances and parameters. First, you must enter a name for your project. The dialogue box above has already been completed and you should enter the information as shown. Then click **Next**.

New Project Wizard

Create Your Opening and Closing Messages

You can enter an opening greeting. Unless you check the box below, VoiceDBC will automatically open with 'Good Morning/Afternoon or Evening' as appropriate but you can add another or an additional message in the box below. e.g. 'Welcome to Joe Pizza Place' or 'Welcome to the Central Banks Account Information Service'. You can also enter a closing message e.g. 'Have a nice day.' or 'Your delivery will take 30 minutes.'

Check to suppress 'Good [time of day]':

Enter an opening greeting:

Enter a closing message:

Help < Back Next > Cancel

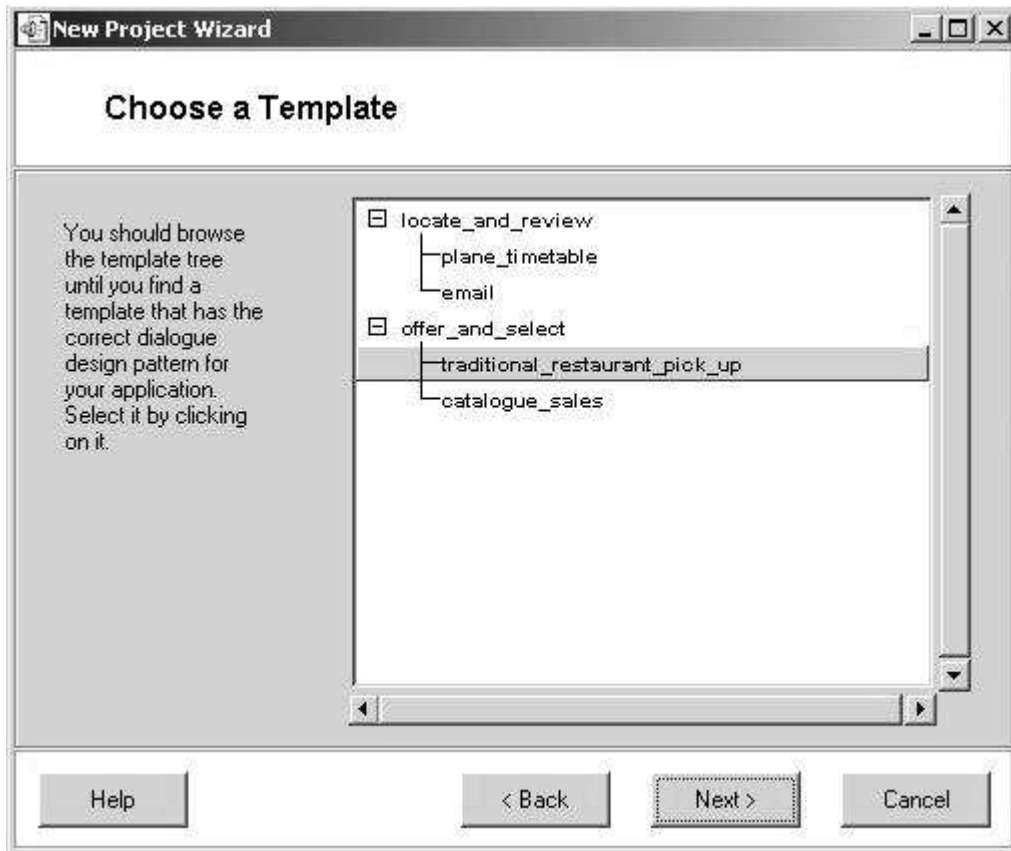
Next, you must specify the opening and closing words for the dialogue. VoiceDBC will say **Good Morning** or **Good Afternoon** as appropriate (unless you check the box to suppress it) but you must add the message to go after those words and a closing message. The dialogue box above has been completed and you should enter the information as shown. Then click **Next**.

Choosing a Dialogue Design Pattern

VoiceDBC is based upon the premise that task-oriented dialogues with a large dialogue-task distance are amenable to being modelled by design patterns. If you are interested in pursuing the theoretical aspects you are referred to Stephen Choularton's thesis - VoiceDBC: A semi-automatic tool for writing speech applications.

VoiceDBC contains a (growing) number of these design patterns. You can always update by visiting www.bymouth.com and downloading the updates. You must choose one appropriate to the domain of your application. In this case click on **traditional restaurant pick up** to select it.

Click on *ext*



CGI and Database Information

Your voice application will undoubtedly require to pass information to a cgi-bin. This may be on your own computer (as it is in the screen-shot) or on a remote computer when it would be characterised as `http://some_computer/cgi-bin/some_file`

In addition, it will require a path to the database (in the dialogue box the path is to the sample of databases shipped with VoiceDBC and is

`C:\VoiceDBC_V1\sample_databases\take_away_pick_up.mdb`) and the name of the table that the application will deal with. The dialogue box below shows a typical setup for a cgi-bin and database on the same computer.

New Project Wizard

CGI and Database Information

(1) Enter the path to the cgi-bin into which you will place your cgi code.

Path to your cgi-bin:

(2) Enter the path to your database.

Database Path:

(3) Enter the name of the table you wish to work with.

Table's Name:

Help < Back Next > Cancel

The Data in the Database

New Project Wizard

Review the Data Generally

VoiceDBC is displaying the fields which make up the table you specified. You have to choose datatypes for them. Click on Analyse to do this. The label will change to Visited when you return to this form.

First you must provide natural language expressions for each field. The database field names are shown in the first column and the default NL references in the data entry boxes in the second column. Overtyp e any you wish to change.

primary_key	<input type="text" value="primary_key"/>	Add Words	Analyse
courses	<input type="text" value="courses"/>	Add Words	Analyse
dishes	<input type="text" value="dishes"/>	Add Words	Analyse
descriptions	<input type="text" value="descriptions"/>	Add Words	Analyse
price	<input type="text" value="price"/>	Add Words	Analyse

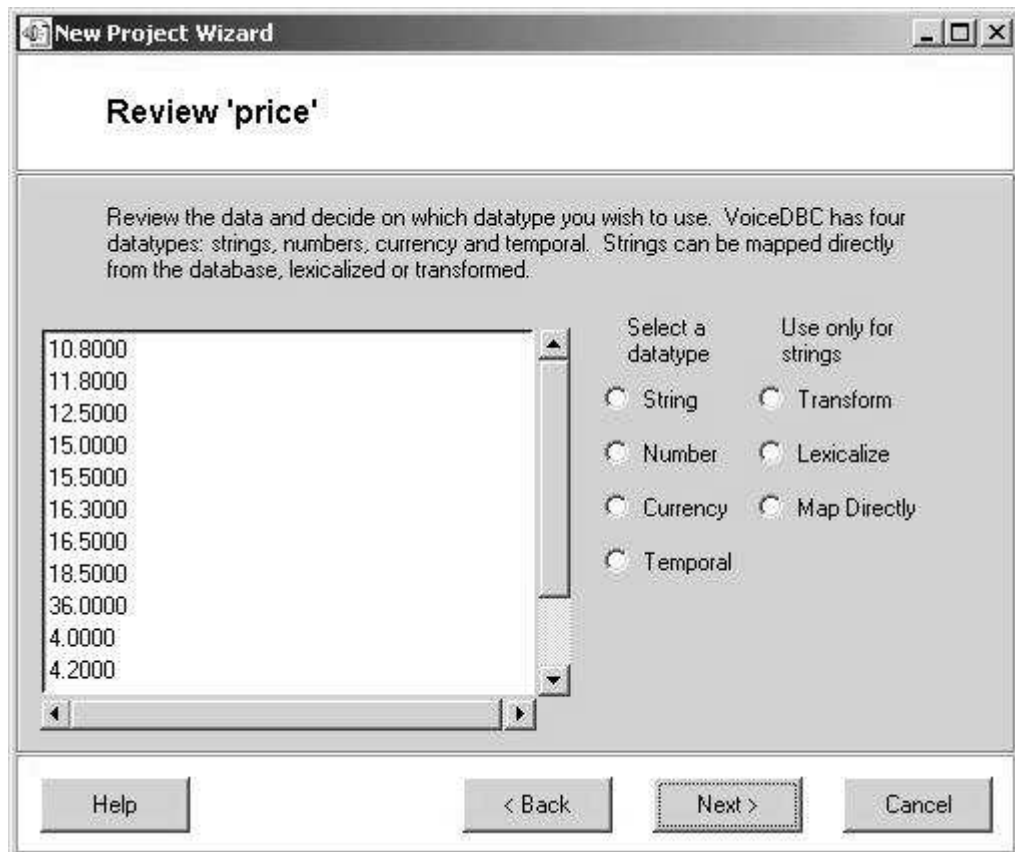
Help < Back Next > Cancel

VoiceDBC offers you the various fields in the database. First you should lexicalise the name by which the database refers to the field. In this case most of them are quite natural

but it would not be unusual to see the departure city in a flight scheduler called from_city as a field in the database. If they require amendment simply over type the existing reference in the white box with the lexical reference you wish to use. VoiceDBC defaults to the field description used in the database (that is the one first shown in this dialogue box) if you do not.

In addition to how one refers to the field you also have to decide how to refer to the data contained in the fields. You have to look at (click on **Analyse**) each one and decide what to do. If you do nothing VoiceDBC will default to simply passing on the data item from the database in the form it originally appears. A token like bri (which might be short for Brisbane) will be passed on with the text-to-speech module to make of it what it can. In the tutorials case many of the data items come from a restaurant menu and do not require transformation but one certainly does. Let's look at price.

Handling the Data



Click on **Analyse** on the price line.

The actual data in the field is deduped and displayed in the text box. You should scroll down and across as required in order to get a feel for the data. VoiceDBC does not rely on any of the proprietary data types used on the different platforms to retain platform independence. All data coming out of a database is expected to be converted to a form suitable for immediate natural language rendering. This is further enhanced through ECMAScript functions. The data you are looking for should be selected as currency and will be rendered in any speech application and 10.800 would be changed to \$10.80. Click on **Currency** and click **ext**.

When you revert to the Initial Data Analysis Form the label on the button by `price` has changed to 'Visited'. Have a look at the other datafields by clicking on **Analyse**. VoiceDBC will default to the form passed by the database but in this case the database entries are in a natural form and do not require further processing.


Click on `ext`.

Data to be Offered by the Application

The screenshot shows a window titled "New Project Wizard" with a sub-header "Choose the Data To Be Offered". Below the header is a text box: "Pick the field to be included in initial system utterances concerning the data, a field (if any) which contains additional descriptive matter concerning the data and a field to group the data by." There are three columns of radio button options, each with a vertical list box above it. The first column is labeled "Initial field" and has options: `primary_key`, `courses`, `dishes` (selected), `descriptions`, and `price`. The second column is labeled "Descriptive field" and has options: `primary_key`, `courses`, `dishes`, `descriptions` (selected), and `price`. The third column is labeled "Grouping field" and has options: `primary_key`, `courses` (selected), `dishes`, `descriptions`, and `price`. At the bottom of the window are four buttons: "Help", "< Back", "Next >" (highlighted), and "Cancel".

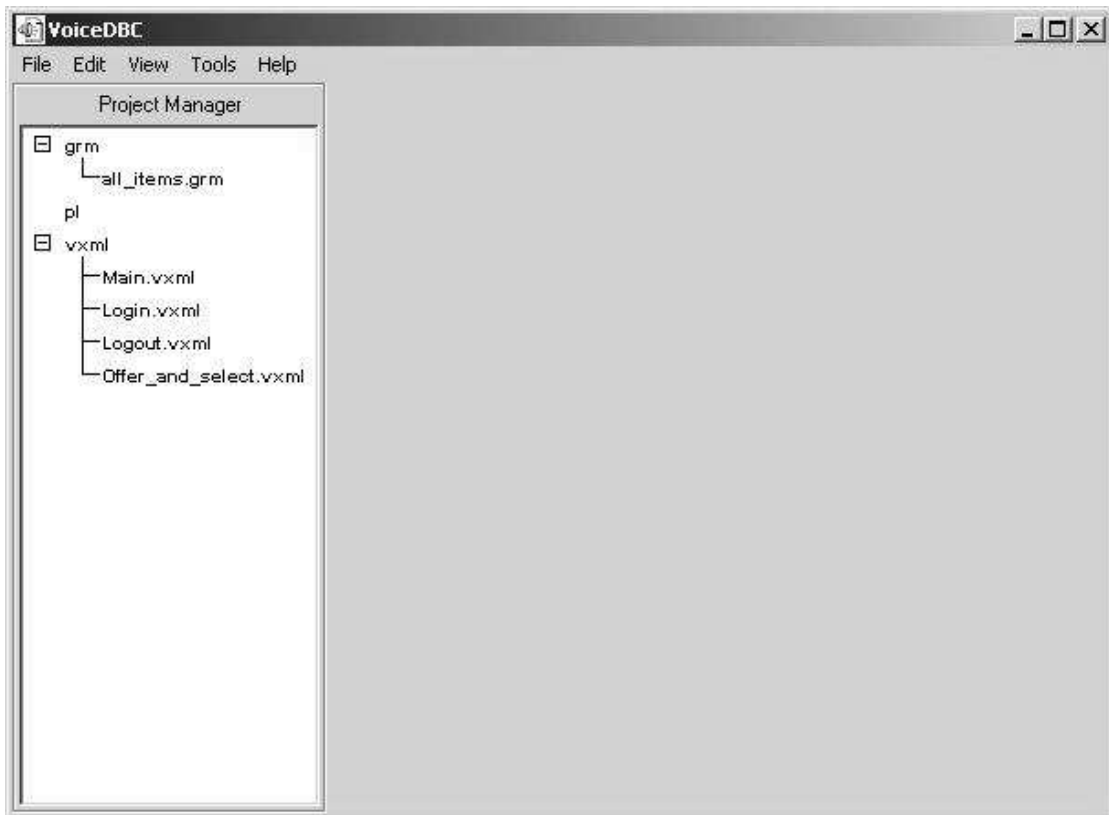
With this type of application the system has to work through a fairly large amount of data. You must choose the focus of the delivery of that data, what additional information should be provided if the user asks for it and how the data should be `chunked` so that the user can skip to a more relevant centre. In the above form `dishes`, `descriptions` and `courses` have been selected for these purposes.

Complete the Application

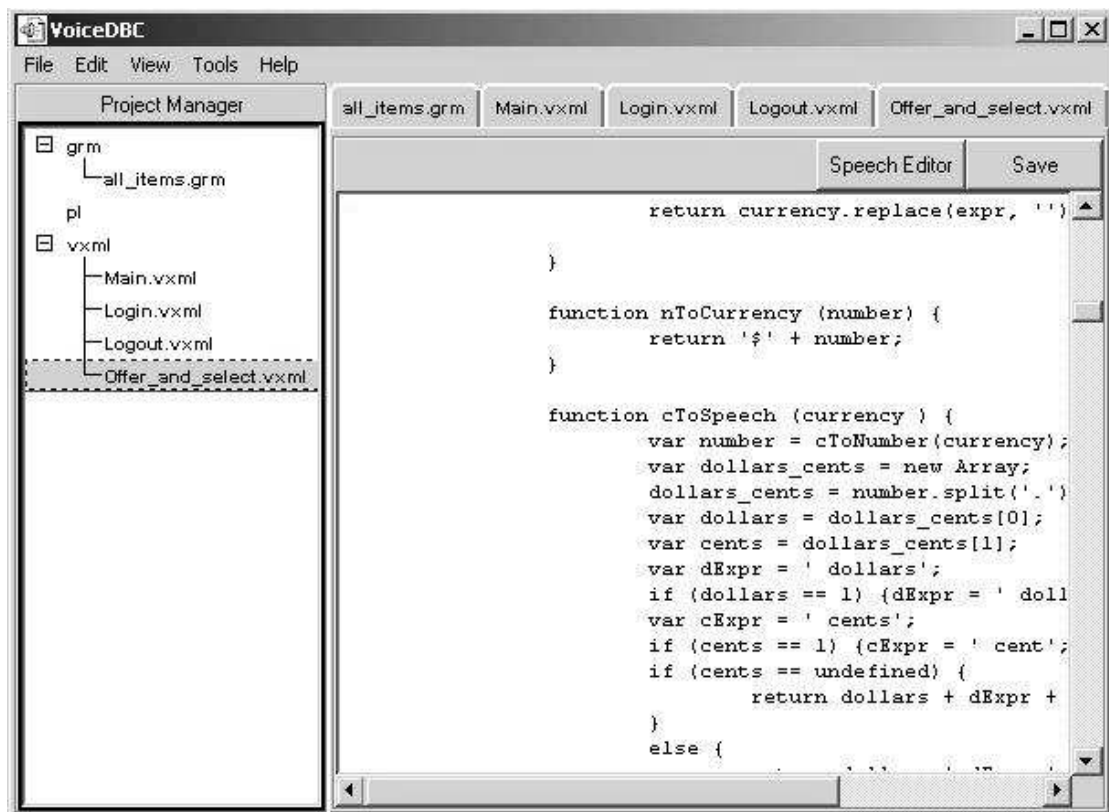
Click on  *ext*.



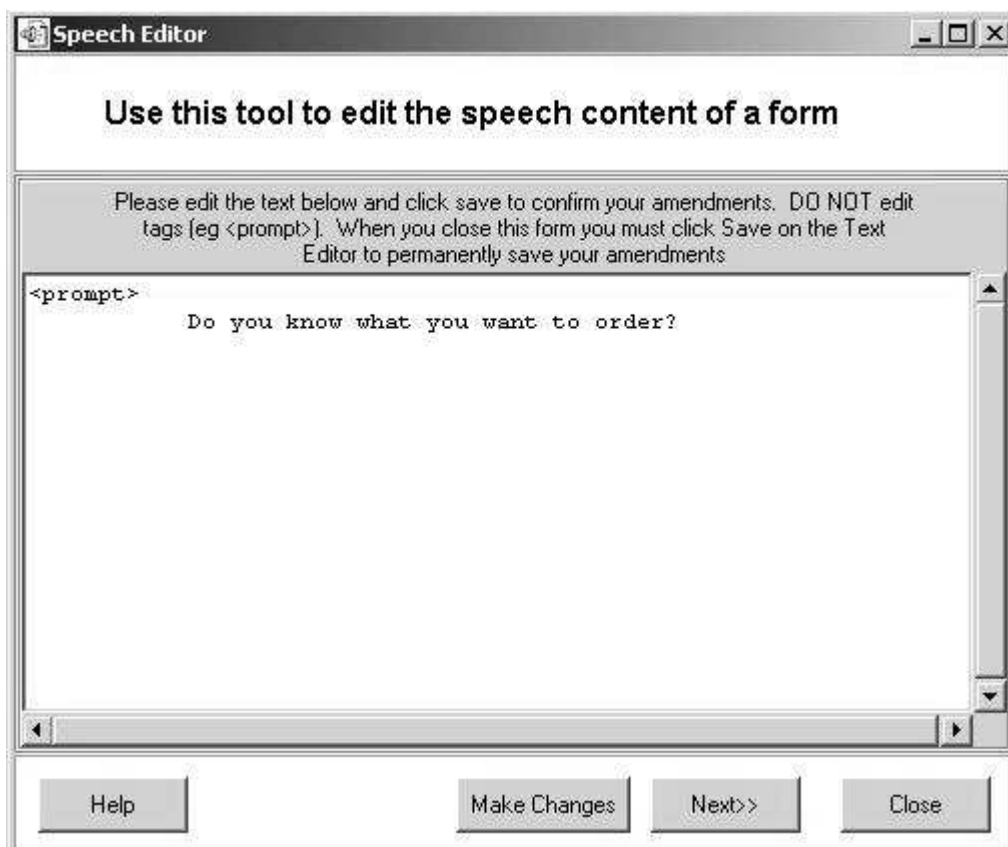
Then click on *Finish*.



Once VoiceDBC has written the various documents required for the application the Project Manager will open displaying the various documents written. Any one can be reviewed by double clicking on the file name. The screen shot below shows all of the documents opened in this way.



Finally the user can review all the system utterance by clicking **Speech Editor** which will work through any system utterances and allow the user to change them if they wish to do so.



You can edit the text that is displayed and click Make Changes if you want to save them. Once you close this form you must also click on Save on the main text editor to save the text file to disk and make the changes permanent.

Deployment

You have to place the VoiceXML (and grammar) documents in the appropriate directories of the voice browser you are using and the Perl scripts into the cgi-bin.

