UNIVERSITY OF GOTHENBURG

Design for a Voice-Based Recipe Book

- From a Communicative Perspective

Author: Xi Chen

Master of Science Thesis in Communication

Report No. 2013:136
ISSN: 1651-4769

University of Gothenburg
Department of Applied Information Technology
Gothenburg, Sweden, May 2009

Abstract

A design of a voice-based recipe book application is carried out in this project. The design is mainly based on a communicative perspective which offers theoretical clues to the human-computer interaction design; therefore, the communication between the system and the users could be more fluent, efficient and similar to a real life interaction. The implementation of the final application is based on VoiceXML and Python, and then tested on the platform offered by Voxeo.

Keywords

Communication Technology, Human-Computer Interaction,

Dialog System, Speech Technology, VoiceXML, Interaction Design

Table of Contents

1. Introduction

Have you ever met the following situations in your life?

- You open your fridge and find that you have onion and eggs, but you don’t know what to cook with them.
- You check the recipe book frequently while you are cooking, which makes you flurried.
- You are so tired and starving after a whole day’s work and want to east something simple and fast to cook.

With the rapid development and widely use of language technology, especially speech technology, it is no longer difficult to deal with the above-mentioned problems by the help of an electronic application – a voice-based recipe book.

In this project, a design of this application is carried out, which is mainly based on a communicative perspective. The reason is that even though it is a human-computer interaction (HCI), it can also be seen as a communication process which is performed between a computer and a human being. Thus, studying the communicative features can help to make the human-computer interaction more fluent, efficient and similar to a real life interaction.

The application is in fact a simple dialog system and will be designed according to VoiceXML standard. Finally, it will be implemented and tested based on the platform provided by Voxeo Corporation. The common gateway interface (CGI) between the database server and the platform is programmed by the program language Python.

The final application is intended to be able to deal with the above-mentioned real-life problems, i.e. it can 1) provide recipes when the user mentions several materials for cooking, 2) read the recipe step by step for the user while the user is cooking, and 3) provide information of cooking time to the user.

This project focuses not only on the final application but also, as mentioned above, on the work of design issues from a communicative perspective. Therefore, the final implementation of the application should integrate both the technique part and the design part (theoretical part).

The following introductions are intended to explain several concepts or notions that are mentioned above.

1.1 Language Technology

No definition of language technology is widely accepted so far. However, the literalness shows that it is closely related to two aspects: technology and language. It is a combination of general linguistics and computer science. “Human language technology” (HLT) and “natural language processing” (NLP) are also mostly used as a substitute of language technology. The boundaries among these definitions are not quite clear, which nevertheless does not affect so much on the theoretical studies and the development of language technology products. Language technology “consists of computational linguistics (or CL) and speech technology as its core but includes also many application oriented aspects of them”[1] such as computational syntax, computational semantics, and so on.

1.2 Speech Technology

“Speech technology relates to the technologies designed to duplicate and respond to the human voice.”[2] The technology is widely used in modern society especially as an aid to the voice-disabled, hearing-disabled, or blind human beings. Many communication aid devices are based on speech technology. Besides, the technology is also used by healthy people, for example, to supplement to computer games and to aid in marketing goods or services by telephone. Two important subfields in speech technology are speech recognition which translates human speech into text, and speech synthesis which artificially produces human speech. Speech-to-text (STT) and text-to-speech (TTS) are commonly used to represent speech recognition and speech synthesis.

1.3 Human-Computer Interaction (HCI)

“Human–computer Interaction (HCI) involves the study, planning, and design of the interaction between people (users) and computers (or other machines). It is often regarded as the intersection of computer science, behavioral sciences, design and several other fields of study.”[3]

To study and to design the user interfaces is always the crucial part,

[1] Language technology, retrieved from http://en.wikipedia.org/wiki/Language_technology on Apr 21st, 2012.
[2] Speech technology, retrieved from http://en.wikipedia.org/wiki/Speech_technology on Apr 21st, 2012.
[3] Human–computer interaction, retrieved from http://en.wikipedia.org/wiki/Human-Computer_Interaction on Apr 21st, 2012.

because the interaction between users and computers occurs at the user interface which includes both software and hardware. In this project, only the software part is concerned.

HCI has a large scope. When studying HCI, both the human side and the computer side are important. On the computer side, techniques such as programming language are relevant; on the human side, communication theories, linguistics, social sciences, and a lot of other fields are relevant. In addition, the design methods and engineering are also relevant.

1.4 Dialog System

A dialog system or conversational agent (CA), which is a computer system aimed to have discourse with a human, with a coherent structure, usually have employed one or more modalities, such as text, speech, graphics, haptics, gestures and other modes, for communication on both the input and the output channel.

In this project, the application is a spoken dialog system, since only speech mode is used. A typical spoken dialog system consists of the following parts: an Automatic speech recognizer (ASR) which is an input recognizer/decoder converting the users' speech into plain text; a Natural language understanding unit (NLU) which is used to analyze the input text; a Dialog manager which is the core component and used to analyze the semantic information; an Output generator which generates natural language output; and a text-to-speech engine (TTS) which is an output render that converts the output into a human appreciable way – speech.

"The dialog manager maintains the history of the dialog, adopts certain dialog strategy, retrieves the content (stored in files or databases), and decides on the best response to the user, and it maintains the dialog flow."[4]

System-initiative dialog, mixed-initiative dialog, user-initiative dialog, and learned strategy are the four staple strategies that the dialog flow can have.

1.5 VoiceXML

VoiceXML (VXML) is a W3C's standard XML format.[5] It is used to stipulate interactive voice dialogues between a human and a computer. Similar to the

[4] Dialog system, retrieved from http://en.wikipedia.org/wiki/Dialog_system on Apr 22nd, 2012
[5] See also: http://www.w3.org/TR/voicexml20/

relationship between HTML and visual applications in which HTML documents can be rendered into visual applications by a visual web browser, a voice browser can render VoiceXML documents into voice applications. Many companies provide voice browsers which are platforms for VoiceXML documents to be executed. These platforms are attached to the Public Switched Telephone Network (PSTN) in order that the interactions between users and the voice applications can be performed over the telephone.[6]

1.6 Voxeo

"Voxeo Corporation[7] is a technology company that specializes in providing development platforms for unified customer experience (self-service) and unified communications (real time communications) applications." [8] The company pays so much attention on standards. Therefore all the platforms developed by Voxeo are based on open standards like VoiceXML, CCXML, and SIP, which makes it easier, efficient and standardized for the development of speech technology products.

1.7 Common Gateway Interface (CGI)

"The Common Gateway Interface (CGI) is a standard method for web server software to delegate the generation of web pages to executable files. Such files are known as CGI scripts; they are programs, often stand-alone applications, usually written in a scripting language."[9]

An example of a CGI program is one implementing a data request from the database server on the VoiceXML platform. The user agent requests a recipe in the VoiceXML platform by saying the dish name; the platform translates the voice input into text and then send it to the server side; the server searches it in the database by the given name, retrieves the information (if it exists), transforms it into VoiceXML standard again, and then sends the result back to the VoiceXML platform.

[6] See also: VoiceXML, retrieved from http://en.wikipedia.org/wiki/VoiceXML on Apr 22nd, 2012
[7] See also the official website of Voxeo: www.voxeo.com
[8] Voxeo, retrieved from http://en.wikipedia.org/wiki/Voxeo on Apr 22nd, 2012
[9] Common gateway interface, retrieved from http://en.wikipedia.org/wiki/Common_Gateway_Interface on Apr 22nd, 2012

1.8 Python

“Python is a general-purpose, high-level programming language used for high-level programming. Python claims to combine ‘remarkable power with very clear syntax’, with a design philosophy which emphasizes code readability. Python's standard library is large and comprehensive.” [10]

2. Literature Review

2.1 Robert Batusek and Ivan Kopecek (1999) in their paper “User interfaces for visually impaired people” discuss specific requirements of visually impaired users, which they think should be crucial to applications and dialogue systems. They believe that the “suitable dialogue strategies” is one of the most important points when building a dialogue system, especially for visually impaired people. They also point out that “in some applications, there is almost no difference in using the user interface between sighted and visually impaired user. This is, for example, the case of dialogue systems that are accessible via telephone.” (Batusek & Kopecek 1999) This, telephone, is exactly the same as the media that is used in this project.

They summarized the specific demands on user interfaces for visually impaired people as following:

1. Comfortable control must be provided by the system via speech and/or keyboard.
2. A speech command dictionary should enable speech commands to be expressed in different ways in order to control the system more intuitively, which is even more important to visually impaired users.
3. The system should enable the user to customize and configure easily depends on how often the system is used and how long has the system been used by the same user.
4. The system need to make sure users should be able to get information quickly and get information overview when needed. This feature can be supported by “various output speech modes and output speech rates as well as by speech summaries, audio glances, earcons and environmental sounds.” (Batusek & Kopecek 1999)

[10] Python (programming language), retrieved from http://en.wikipedia.org/wiki/Python_(programming_language) on Apr 22nd, 2012

5. The information about the position should support the orientation of the user. And this feature can be supported by the same ways as last point.

All sound required in the whole system consists of three types: 1) synthesized voice generated by a voice synthesizer, 2) sample voice such as recorded human-voice, and 3) non-speech sound produced by the sound synthesizer, MIDI, wave tables or special samples. Using variety types of sound could enhance the efficiency and effectiveness of the communication between the system and the user.

However, they can also make the user confused sometimes. Thus, the authors pointed out some strategy to optimize this problem:

It’s better for the system to detect or receive declaration from the user about whether it is a beginner or an experienced user. Then, the system decides a corresponding strategy depending on the user experience measure. The user should be able to ask for an explanation of the meaning of the implicit information anytime during using. If the user asks for explanation frequently, the system should be aware of that and then shift to another communication level. At last, the user should be able to choose between communication levels freely.

2.2 In “VoiceXML-based spoken language interactive system” (Ondáš 2006) the author Ondas introduced the architecture of spoken language interactive system (SLIS). The general model of SLIS is firstly introduced in the paper, which includes the following parts: 1) Input-output block (I/O) which processes the input and output speech signal, 2) Automatic speech recognition (ASR) which recognizes input speech from user and convert it to text, 3) Natural Language Understanding (NLU) which analyzes the meaning of converted recognized words and generate the meaning in the system’ s context, 4) Dialog Manager (DM) which controls all operations in the system, 5) Natural Language Generation (NLG) which generates representation of information in order to respond to the user, and 6) Text to speech (TTS) which convert text to phonetic information to the user. The authors also introduce two different types of architectures of SLIS, pipeline architectures and distributed architectures. The former is old, strictly ordered, low flexible and has late responses to errors, while the latter is more modular and has high ability to process management and is flexible to add new features.

2.3 Kristiina Jokinen (2004), in her paper “Communicative competence and adaptation in a spoken dialogue system”, emphasizes that when designing a practical system which can function. The author introduces a new metaphor for human-computer interaction: “the computer is not only a tool that the user must learn to use, but it is a (software) agent interacts with the user and mediates between the user and the application.” Thus, interactive systems need not only to communicate with user but also to satisfy a bigger range of users. This is why user study is very important.

Kristiina then describes three common approaches regarding the integration of human aspects into interactive system design: user-centered design, ergonomics and user modeling.

User-centered design which focuses on the quality of interaction requires user to participate in the design process and experiment the prototypes and also give feedback of how the system should be better changed. This approach links to the context of use, context of organizational, technical and physical factors.

Ergonomics focuses on applying scientific information from anatomy, physiology and psychology into design, so that environment, user capabilities and limitations are considered. This approach helps to design, for instance, the keyboard, displays and also take user’ s cognitive load into account. The interface design, especially system prompts, can benefit from this approach, so that the system prompts properly and unambiguously, and the user could get correct information about the task and know the system’s ability and limitation and thus give an appropriate answer back to the system.

Studying human communicative features, such as speaker intentions, speech and language errors, mutual beliefs, etc. and building models from single users for their computational treatment can help to better the human-computer interaction. User models can be both simple lists which show user preferences, and complicated models which are usually related to system adaptations, which means that the system has the ability to detect user behavior and learn it.

All three approaches focus on and result in differently, but all highlight interaction between the computer and the user. They author pointed out that the adaptation happens in the user part in order to get accustomed to the system. However, the system should also be capable to detect and adjust to the user’s behaviors. This can be explained in this way: if we think about natural language communication, mutual adaptation is the essential communicative capability of each participant in the conversation.

The author then illustrates the communicative competence which a communicatively adequate, user-oriented system should have are "physical feasibility of the interface, efficiency of reasoning components, natural language robustness, and conversational adequacy". (Jokinen 2004) 1) In the system interface, enablements for communication ought to exist, so that the system is easy to use, learn, and talk to, and system ability and restrictions should be transparent to the user. 2) User tends to think that the human-computer communication is different from natural language communication, so unnecessary pauses can exist, which could influence the efficiency or cause a conversation termination. Thus, efficiency of reasoning components is a fundamental communicative competence of the system. 3) Natural language processing (interpretation and generation) is essential to ensure elaborated interactive system and their communicative capability, even though people may think task accomplishment is more important and the vocabulary is limited in human-computer conversation. 4) The system should be capable to deal with ambiguousness, confusedness, misunderstanding, etc. provided from the user side, and convey useful collaborative information to the user in order that all underlying tasks could be done.

Regarding evaluation, the author points out that both system design and the perception of the user should be considered, because both system efficiency, error-free processing, etc. and user's comfort caused by the system design are crucial to user satisfaction and should be evaluated. To evaluate the adaptation is important but complicated, which calls for a longer time.

2.4 David Sadek (1999) describes in his article "Design considerations on dialogue systems: From theory to technology –The case of Artimis-" that because of the emerging amount of human-computer interaction on modern media and the nature and fundamentality of speech as the way of communication, especially the speech as the only modality to telephone users, speech technology has been considered as an appropriate and emerging media for human-computer interaction. User-friendliness is very crucial to an interaction and system's intelligence is also fundamental. However, it is hard to balance the two. For example, it is hard to get wanted information for the user through a single query or the answer from the system could be of very big size even if the user's query is well-formed. Thus, clarification, negotiation and completion dialogues are required to complete a certain task. Furthermore, in order that the user could express more freely, the agent's intelligence and knowledge are very important.

The author has provided several features which a user-friendly system should embody:

Negotiation ability can enhance the system's user-friendliness under three circumstances. 1) The user's request to the system is incomplete hoping that the system could help to express more specific searching conditions. 2) The system generates a big size of answers according to the user's well-formulated request. Thus, the system should help the user to constrain the results. 3) The system cannot find any appropriate answers according to the user's request, so the user should be directed to revise the searching conditions and formulate searchable ones.

Contextual interpretation and language flexibility enable the system to understand the user's meaning with respect to what the user has already said before, so that the user need not to express a well-formulated request every time under a certain task, because people tend to express only the differences between situations rather than a whole situation. The system should also support language flexibility because the user could choose words from a large vocabulary and speak spontaneously when describing the difference between situations.

Interaction flexibility means that the system follows no pre-established interaction structure, so that the user has the possibility to "engage into a clarification sub-dialog before answering a system's question, or to change topic or dialogue objectives even before the completion of a mutually agreed dialogue task." (Sadek, 1999)

Cooperative reactions enable the system to provide 1) completion answers, 2) corrective answers, 3) suggestive answers, 4) conditional answers, and 5) intensional answers when interact with the user. For example, when the user asks "Do you know when the train leaves", the system should give the exact time rather than saying just "Yes, I do."

Adequacy of response style provides the system with ability to show the answers to the user in different modalities appropriately, i.e. the system should be able to coordinate among different medium according to the current situation.

The author points out that these criterions are interdependent and should be fulfilled together and generically. Thus, when designing a dialogue system, a global approach should be carried out so that all criterions must be considered.

The author then illustrates two types of computational approaches for dialogue system. **Structural approaches** are derived from either a computational background or a linguistic background, which can be represented by a finite-state automation or a CFG (context-free grammar). They assume that in all dialogues or interactions, there are regularly structured frameworks. Thus, they are rigid and limited to the user. While **Classical plan-oriented approaches** assume that all communication interactions are comprised of not only a set of words but also a series of communicative acts, for example, request, feedback, confirm, etc. And all these actions are motivated by achieving a specific goal in a certain interaction between interlocutors whose mental states (for example, beliefs, intentions, etc.) would be changed. The purpose of an interaction is to acknowledge the goals and plans which people already generally have in their mind, and to generate effects which are accordant with the original purpose.

2.5 Allwood and Ahlsén (2009) in their article point out that the rapid development of intercultural information communication technology has enlarged the use of multimodal intercultural communicators. This trend, on the other hand, reflects the requirement that communicators master natural communication abilities, so that people with lower communication skills or low education could also use.

Activity features must be considered when designing and evaluating, since different activity requires differently, regarding its purpose, roles, artifacts, environment, etc.

All communicators should be based on "a generic system for interactive communication with a number of parameters that can be set to capture intercultural variation in communication" (Allwood and Ahlsén 2009), i.e. a generic multimodal intercultural communicator (GMIC).

Moreover, communicators should have the capability to deal with recognition or understanding collapse, so that the system should run meaningful process to handle the problem rather than just reporting error.

The authors have also illustrated an outline of some intercultural parameters which should be considered when designing, including cultural variation in expressive behaviors (head, eyes, arm, and shoulder movements, gazing, etc.), in content and function (emotions, attitudes, everyday topics, and common speech acts), in perception, understanding and interpretation. There are also some interactive features that worth to consider: turntaking, feedback, sequencing, and spatial configuration. In addition, some other contextual features (for example, use of pronouns and tense endings may not exist in other languages; beliefs and values could also influence communication) and other aspects of behaviors must also be examined.

3. Theoretical Background

As mentioned above, quantities of theories involved in different fields are worth to consider when designing this application. However, only communication theories are focused in this project, since HCI should essentially be seen as a communication process and few studies have connected design works of dialog systems with communication theories.

3.1 Patterns of Communication

The concept of “patterns of communication” brought out by Jens Allwood is “fairly general and does not imply very much more than repeated traits of, or aspects of the communication of the members of a certain social or cultural group.” (Allwood, 1999, p.1) The concept is mostly used in the studies of intercultural communication. However, a lot of notions are quite useful in this project if we see human-computer interaction as the communication between two different groups: human and computer. Allwood puts forward a framework for the study of spoken language communication which can also be regarded as an operational analysis of the concept of “patterns of communication”. This design project is mainly guided by the items in his framework. All the notions which are related to this project are listed and explained in the following. Some notions such as spatial arrangements and nonverbal behaviors are omitted in this part since they are irrelevant to a voice-based application.

The first three – typical sequences of events, turntaking, and feedback – are patterns of interactive communication between speakers and listeners, and the following three – phonological patterns, vocabularies, and grammatical patterns are patterns in individual communicative behaviors

3.1.1 Typical Sequences of Events

Typical sequences of events refer to that many conversations start, move ahead and end in certain sequences depending on cultural, convention, formality, place, time, artifacts, modalities and so many possible factors. For example, how we start and end a telephone call must differ from how we start and end in a seminar: we do not say “This is Rose speaking. Who’s that” to start a seminar or “Welcome to this…” at the beginning of a phone call either. The way of how we greet, introduce people, say goodbye, etc. are almost habitual, regular, and conventional.

3.1.2 Turntaking

Turntaking refers to “the distribution of the right to speak which is related to such phenomena as how long one can speak and if one can speak simultaneously with other speakers.” (Allwood, 1999, p.3)

Turntaking strategies vary among the cultures of the world. In some cultures, people speak simultaneously with others to compete for attention directly. However, in most other cultures, especially those which have low tolerance for interruption, people draw attention indirectly by competing for the floor rather than interrupting people directly. In some situations, interruption seems as insult.

The period of time between speakers that values the right to the floor also differs from cultures. For example, the pause in USA is longer than in Sweden and in Finland it is even shorter.

It is also interesting to examine how one signal that he or she wants the floor, keeps the floor, and is ending the floor. For example in China, people sometimes repeat the word which sounds like “nai-g” to show that we are hesitating or thinking but we still want to maintain the floor. If we want to end the floor, we can gaze at another person to give out the floor.

3.1.3 Feedback

Feedback refers to “the fact that speaker as well as listener, in a conversation must know how the other party is reacting.” (Allwood, 1999, p.3) During a conversation, it is significant for both the speaker and the listener to make sure whether they are perceived and understood by the other party correctly and they perceive and understand the other party correctly through the whole communication process; otherwise, the communication will be ineffective and inefficient or even a communication failure will arise.

Both parties can act as feedback elicitor. For example, in English, people can use a tag-question, such as “you want to cook Swedish meatballs, do you” and people can also say “do you mean that you want to cook Swedish meatballs".

Both parties can act also as feedback giver and the most common way to give a feedback, for example in Chinese Mandarin, is to either nod your head or to utter an "eng" sound with a falling tone or to say "duei" which means "correct" in Chinese.

How much feedback information to give is very important but also depends on culture. Incomprehension arises when feedback giving is less than expected, while giving more feedback than expected leads to ridiculousness, annoying and irritating.

3.1.4 Phonological Patterns

Phonological patterns mean the sounds which are used when one speaks. The properties of both isolated sounds and longer sequences of sounds are worth to pay attention; these properties are, for example, intonation, stress and melody. Sound which has numerous functions, for instance, a funny sound or a boring sound, a rising tone or a falling tone, can arouse people's emotions and indicate whether it is a statement, a question or an imperative.

3.1.5 Vocabulary

Vocabulary refers to the words and expressions that are used in different types of communication. Vocabulary variation is influenced by cultures and nations and even by professions and activities. For example, there are more words referring various kinds of peppers in Chinese than in English; eggplant used in American English while aubergine is used in British English, but they refer to the same kind of vegetable.

3.1.6 Grammatical Patterns

"Grammatical patterns refer to the differences in word order and types of linguistic construction which exist between different languages and between different ways of using a language." (Allwood, 1999, p.3)

3.1.7 Interpretation and Understanding

Interpretation and understanding means that having the ability to speak is probably not enough for people to communicate; besides, one must be able to understand and interpret what the other people express. One need pre-stored information, especially culture specific background information, with which he/she can connect other people's nonverbal messages, words, phonological patterns and grammatical patterns. The more pre-stored information one has and the more relevant the pre-stored information is, the easier interpretation and understanding will be.

3.2 Other Relevant Theories

3.2.1 More about Feedback

Feedback is the backbone of communication. If there is no feedback, the communication will break down easily. In face-to-face communication, feedback is inevitable in communicative actions and expressions. The expression of emotions and attitudes are included as a part of feedback as well. Thus, feedback is the interaction between two people where reaction can be shown from the interlocutor in terms of attention, understanding and agreement.

"Feedback and eliciting are annotated by means of the same sets of attributes, called Basic, Acceptance, and Additional emotion/attitudes" (Allwood et al., 2005, p.4).

Allwood describes how the feedback giving is practically coordinated as following:

Table 1

Function attribute		Function Value
FEEDBACK GIVE	Basic	Contact/continuation, Perception, Understanding (CPU) Contact/ Continuation Perception (CP)
	Acceptance	Accept Non-accept
	Additional Emotions/Attitudes	Happy, Sad, Surprised, Disgusted, Angry, Frightened, Certain, Uncertain, Interested, Uninterested, Disappointed, Satisfied, Other

Basic function attribute regards to the interaction between speaker and listener(s) whether they have a desire to acknowledge contact (C) and perception (P) of each other. After that the interlocutor will indicate understanding (U) message by eliciting a sign of understanding – verbal and nonverbal communication – such as “Huh,” “Umm, I see” or head nodding.

The function attribute **acceptance** indicates that the message has been understood, perceived, agreed on or denied by the interlocutor; for instance to give the code of agreement words: “Yes”, “Sounds good”.

Emotions and attitudes are automatic features which can occur simultaneously with basic feedback, acceptance feature.

4. Design

Based on the theoretical introductions above, the analyses and the design works are included in this part.

4.1 Main Functions

To search for a recipe by the name of a dish;

To search for a recipe that can be cooked by the materials the user has in the fridge;

To provide dish introductions when more than one result are found, in order that the user can choose which one to cook;

To read the recipe step by step for the user and jump between steps while the user is cooking.

4.2 Sequences

4.2.1 Greeting

Even if the application will be developed and tested on the Public Switched Telephone Network, it is still supposed that the system could be installed in other devices in the future, which is located in the kitchen as a cooking assistance system and easy to access and use. Therefore, conversation starts as if it is an assistant rather than a regular phone call. So, the initial sequences are like this: The system will wake up from a sleeping mode and say “Hello! Welcome to use the cooking assistant” when the user calls “cooking assistant”, and then the system will ask “what can I do for you” as people do in a shop or a service station.

```
<noinput> </noinput>
<nomatch> </nomatch>
<form id="wakeup">
      <field name=”wakeup”>
            <option>wake up</option>
            <option>cooking assistant</option>
            <filled>
                  <prompt>
                        Hello! Welcome to use the system!
                        What can I do for you?
                  </prompt>
            </filled>
      </field>
</form>
```

4.2.2 Medial Sequences

When we cook with the help of a paper-based recipe book, we usually firstly check by the dish name or by the main material such as chicken, pork, etc. Second, we choose one from all the matched recipes after reading the descriptions of them. At last, we cook according to the steps showed in the recipe.

The medial sequences in this application will mostly imitate the sequences as mentioned above, but there are surely a lot of things changed according to the technical limitations of VoiceXML and the particularity of human-computer communication.

4.2.3 Leavetaking

Fewer leavetaking seems better since after cooking a dish, the user has no more interest in talking with the system rather than eating, and so, if the user has no more demand from the system, one sentence “have a good appetite” is enough for the leavetaking.

```
<form id="leavetaking">
      <field name=”morehelp” type=”boolean”>
```

```
                <prompt>
                     Cooking is finished. Do you need more help?
               </prompt>
                <filled>
                    <if cond="morehelp”>
                          <submit next=”command.py "/>
                    <else/>
                          <prompt>OK! Have a good appetite!</prompt>
                          <exit/>
                    </if>
                </filled>
          </field>
   </form>
```

4.3 Turntaking

4.3.1 Interruption

There is a property in VoiceXML called “bargein” which is used to enable or disable users’ interruption when the system is prompting. When the value of “bargein” is “true” the system’s prompt can be stopped by the user’s utterence; while the value is “false”, no sound could stops the system, i.e. the system will not stop talking until the end of the prompt.

When the “bargein” value is “true”, another property called “bargeintype” becomes useful. There are two values that could be chosen: “speech” which means that the prompt will be stopped as soon as any sound (including background noise) is detected irrespective of whether or not the input matches a grammar; and “hotword” which means the prompt will not be stopped until a complete match of an active grammar is detected, i.e. input which does not match an active grammar will be ignored and the prompt will not be interrupted.

So, the best way to deal with interruption is to set the “bargeintype” value into “hotword” and also set as a global value. Therefore, during the whole conversation, the system cannot be interrupted by background noise, but the users who are familiar with the system and do not want to wait until the prompt is finished can get their turn to talk directly by saying expected words which match the grammar.

In order to set this global value,

```
<property name="bargeintype" value="hotword"/>
```

will used and placed in the beginning of the code.

4.3.2 Duration between Turns

The period of time between turns can be assigned by the “timeout” property. In other words, if the user does not respond anything after the assigned period of time followed by the system’s prompt, the system will throw a “noinput” event.

In order to make sure that the user has enough time to think before respond or not to waste so long time here in case the user has no idea about what to respond, it is better to set the “timeout” value to “5s” (5 seconds).

Therefore, if the user does not know what to say, the system can take the turn again, re-prompting or doing something else depending on what is the “<noinput>” element.

```
<prompt timeout=”5s”>
      Do you know the name of the dish which you want to cook?
</prompt>
<noinput count=”1”>
      <reprompt/>
</noinput>
<noinput count=”2”>
      <prompt>
            Please say yes if you know the dish name which you want
to cook. Otherwise, say no.
      </prompt>
</noinput>
```

4.3.3 Signaling

In human-computer communication, it is not easy for any party to signal when wanting, maintaining or ending a turn.

A user can say “hotwords” to signal when wanting the turn; however, as mentioned before, he/she should have already got the turn when the “hotwords” are detected. A user can leave a silence when signaling the end of the turn. Unfortunately, a user has to talk without long silences in order to maintain a turn.

The system usually signals the end of the turn by saying a full sentence from which the user should be able to perceive that the system is ending its turn and he/she is expected to give a respond. The system does not need to signal for maintaining a turn or cannot ask for a turn either.

Thus, it is better for the system to use fewer sentences in a turn and better to say a question or an imperative. The expressions in the sentences should be easily understandable in order that the user knows exactly what is expected to respond without hesitating during his/her response turn. In addition, the system should ask simple questions rather than difficult ones, so the user can say only a few words or shorter sentences which can minimize the risk of causing pause when responding.

4.4 Feedback

Due to the platform limitations, the system can neither give feedback through non-verbal communication, such as eye contact, gestures or head-nodding, nor give verbal messages at the same time when the user is speaking. However, it would be better if the system somehow give feedbacks after the user has finished talking.

Giving feedbacks to users are crucial in this application which has several merits. First, it makes the user feel the continuation and that the conversation is as real as talking to a person. Second, since the environment factors are precarious which could be noisy or bad quality, it is necessary to confirm what information is received from the user, sometimes by repeat the user’s sentence; this step is to show the continuation, perception and understanding all together. Third, feedbacks can show acceptance or non-acceptance to the user, for example, the system says “sorry, I don’t understand. Please try again” if what the user says does not match the grammar. Last, the system can even show a little emotion through feedback giving, for example, the system says “sorry! No dish is found in the database” with a stress sound on “sorry”.

In this application, it is even more important for the system to elicit feedbacks from the user. As stated above that the system often need to confirm whether the understanding is correct, thus, the system has to elicit a feedback which requests a confirmation from the user side. For example, “I heard you want to cook Swedish meatballs, don’t you?” “So, you want to cook with potatoes and butter, is that right?”

Nevertheless, if the whole conversation is filled with this kind of feedback giving and eliciting, it seems tedious and iterating, which could drive the user crazy and cause dissatisfaction.

In the initial section, if the user is not familiar with the system, he/she may not know what to say or say wrong words which do not match the grammar, when the system asks “what can I do for you”. In this case, if the system detected nothing valid feedback from the user after few seconds, the <noinput> or <nomatch> event will be activated in which the system will say some introductory words to tell the user what to say. If the user provides no valid feedback for a second time, the system will redirect to a close question mode,

```
<noinput count=”2”>
      <goto next=”#closequestions”>
</noinput>
```

which asks the user “do you know the dish name that you want to cook?” This can effectively avoid the communication failure caused by wrong feedback.

4.5 Phonological Issues

Not so much phonological factors can be considered, since the platform does not support well. However, there are still some points which need to be brought out.

The exclamation mark can be used somewhere in the conversation, which generates emotional feedback to the user, for example, “have a good appetite!” and “hello! Welcome to use the system!”

```
<prompt>
      Have a good appetite! Bye!
</noinput>
```

It’s better to use full sentences rather than short ones when it is a question, because the TTS system, in most cases, generates same intonation no matter it is a full stop or a question mark. For example, to use “is that right” followed by a question mark instead of “right” followed by a question mark, because in the output sound, “right” followed by a full stop has no difference with “right” followed by a question mark. Thus, the users can be confused when the system just says “right” with a falling intonation.

Due to the current low-developed TTS technology, a problem which causes so much inconveniences and troubles to the end users and lows down the system’s usability is that when the system reads long sentences located in the cooking steps and the ingredients, the words speed is so fast which causes unclearness and difficulty to understand. One way to deal with this problem is to add pause between the words in order to separate the important words, either by add “<pause>” tag or by add comma or full-stop.

```
<prompt>
    So, I heard that, you want to eat, Swedish meatballs. Is that right?
</prompt>
```

4.6 Vocabulary and Expressions

When writing the grammar file, it should be as inclusive as possible, such as “I want to know how can I cook ***”, which were considered as not necessary by the other members because no users will express like that. However, every possible expression that the users could say should be considered. It will be useful even if very few people will say like that. The system should be considerate for all users who are both English natives and not native speakers. English native speakers usually use short, simple, oral words, preposition phrases, and even slangs, while non-native speakers tend to use normal, written, regular words. For example,

```
<rule>
    I
    <one-of>
        <item> want to </item>
        <item> wanna </item>
    </one-of>
    <one-of>
        <item>
            <one-of>
                <item> cook </item>
```

```
                    <item> know how to cook </item>
                    <item> can I cook </item>
                </one-of>
            </item>
            <item>
                eat
            </item>
        </one-of>
        <ruleref uri="#dishname"/>.
    </item>
```

The variety of using vocabularies in differences cultures should be considered. The possible way is to stored all the possible words in the materials' database no matter it is used in British English or American English, or even other languages but having no direct translation into English. The words which have the same meaning should be connected in order that the system knows exactly what it is when for example one user says "aubergine" and the other says "eggplant". (cf. 5.4.3 Material Table)

Sometime, the users who are not English native speakers may have difficulties to distinguish countable noun and uncountable noun, may not accurately use the plural form and single form of the materials, or may wrongly uses an article in front of the noun. However, it is not difficult to deal with. Both "a" and "an" and both single and plural forms of the words could be put into the grammar, i.e. no matter the user says "two eggplant" or "a eggplant", they could all be matched to the grammar.

Sometimes, the users could say "I have a bottle of milk" rather than "I have milk", or "I have some potatoes" rather than "I have potatoes". Thus, for the system to be more user-friendly, a list of qualifier words should be included in the grammar.

```
<rule id="determiner">
    <one-of>
        <item></item>
        <item>a</item>
        <item>an</item>
        <item>some</item>
        <item>several</item>
        <item>a few</item>
```

```
            <item>many</item>
            <item>a lot of</item>
            <item>plenty of</item>
            <item>a little</item>
            <item>one</item>
            <item>two</item>
            <item>three</item>
            <item>four</item>
            <item>five</item>
            <item>handful</item>
            <item>a bottle of</item>
            <item>a can of</item>
        </one-of>
    </rule>
```

In addition, there is no need to put too much consideration on the irregular forms of turning words into plural, because the voice recognition system is not so strict, for example, the word “tomatos”, “candys” and “peachs” can be successfully recognized by the system, even though the correct forms should be “tomatoes”, “candies”, and “peaches”.

```
# Open a database connection
con = sqlite3.connect('recipes.db')
con.text_factory = str

# Create a cursor
cur = con.cursor()

# Retrieve data from material table
cur.execute('SELECT * FROM Ingredientstbl')
materials = cur.fetchall()

# Create materials list with both singular forms and plural forms.
material_list = ""
for i in materials:
        material_list += '<item>%s</item>' % item[0]
        material_list += '<item>%ss<tag>out="%s"</tag></item>' % (i[0],i[0])
```

Another problem is whether to enable the users to search by both main materials and side materials or just by main materials. It is hard to identify main materials and side materials and it is also an intercultural problem since people having different culture background may hold different opinions towards materials and ingredients. In order to be considerate for all end users supposing they are from all over the world, thus, all names of materials should be included in the grammar as long as they appear in the database. (cf. 5.4.3 Material Table)

4.7 Dialog Sample

According to the analyses above and the main intended functions, five prototype dialogs are integrated in the following table. Dialog 1 and 2 perform the database search by cooking materials while dialog 3, 4 and 5 are based on searching by materials. Dialog 5 also includes the function of adding conditions. Dialog 1, 3 and 5 use open questions while dialog 2 and 4 use close questions when the user fails to answer an open question. The discourse with GREY background is given by the system and that with WHITE background is given by the user.

Table 2

1. Open	**2.** Close	**3.** Open	**4.** Close	**5.** Open
Search by dish name		Search by materials		Search by materials + Add condition
Cooking assistant!				
Yes, what can I do for you?				
How to cook Swedish Meatballs ?	Err...	What can I cook with beef and flour?	Err...	What can I cook with beef?
	What can i do for you?		What can i do for you?	So, you want to cook with beef. Four dishes have been found. Do you want to continue?
	Err...		Err...	
	Do you know the name of the dish you want to cook?		Do you know the name of the dish you want to cook?	
	Yes		No	Add a condition.
	What's the name of the dish?		What materials do you have? Name one or two.	What do you want to add?

Swedish meatballs.	I have beef and flour.	Cooking time no more than 90 minutes.
I heard that you want to eat Swedish meatballs. Is that right?	So, you want to cook with beef and flour. Two dishes have been found. Do you want to continue?	So, you want to cook with beef, in less than 90 minutes. One dish is found. Do you want to continue?
Yes	Yes	Yes
	Here are the options: 1. Swedish Meatballs. 2. Beef pie. Which one do you prefer?	The dish is called Swedish meatballs. Do you want it?
	Swedish Meatballs (Number One).	What's Swedish meatball? Swedish meatball is a dish which...... Would you like it? Yes.
Do you want to listen to the materials of Swedish meatballs?		
No	Yes Main materials: ...... Side materials:	
The whole process needs around 80 minutes. Do you want to cook now?		
Yes.		
OK. Let's starting cooking. There are totally 11 steps to cook Swedish meatballs. Please follow the steps carefully. Step 1: Beat egg and add milk and bread crumbs.		
Go back. This is the first step. Next step.	Next step.	Repeat. Step 1: ...... Next step.
Step2: Combine crumb mixture with cooked onion, salt, black pepper, ground beef, and ground pork.		
Go back. Step1: ...... Step 3.	Next step.	Repeat. Step 2: ...... Step 3.
Step 3: .... (from step 3 to step 11) Step 11:		
Go back. Step 10: ...... Next step. Step 11: ...... Next step.	Next step.	Repeat. Step 11: ...... Next step.
This is the last step. Cooking is finished. Do you need more help?		
Yes. OK. You are going back to the main menu.	No. OK! Have a nice appetite!	

5. Implementation

In this part, issues concerning how the application is finally implemented are presented. First, how the application is divided into sections is introduced. Second, the flowcharts of the dialogs of each section are showed. Then, the files that are included in this application are listed and are described regarding the usage. At last, how the database is organized is demonstrated.

5.1 Sections

The whole application has three sections in general: the initial section in which greetings are given and search conditions are collected from the user; the recipe selection section which is in charge of providing information about recipes in the searching result and enables the user to choose; the cooking guide section which provides cooking instructions step by step.

5.1.1 Initial Section

The system greets the user first and asks “what can I help you” which is an open-question. The user answers with the search condition by mentioning either a dish name or a series of materials that he/she wants to use for cooking. If the user failed to provide grammar-matched answers, the system will offer support instructions and if this happens again, the system will turn to close-question mode and asks whether the user knows the dish name. If no, the system will ask the user to mention some materials that the user has. Both open-question mode and close-question mode go to a confirmation where the system will repeat the conditions given by the user, and tell the user how many dishes can be found, and ask if the user wants to continue. If the user says no, the dialog will start from the beginning. If the user says yes, the dialog will continue to the recipe selection section. The user could also say “add a condition” after which the system will ask “what more materials do you have?” After the user adds conditions, the system will redirect to the confirmation step again, performing a loop. At last, the dialog will come to the second section – recipe selection.

5.1.2 Recipe Selection Section

If there is only one search result, or if the user mentions a dish name directly in the previous section, the system will repeat the name of the dish and ask whether the user wants to listen to all the materials that are needed in the following cooking process. If the user says yes, the system will read and then asks if the user wants to listen again in case it is too long or too fast for the user to check; if the user says no, the system will tell the user the approximate cooking time and ask for a confirmation of whether the user wants to continue to cook. Here, the user could say yes to continue, say no to start over from the initial section. In the beginning of this section, the user could ask could also ask, for example “what is Swedish meatball”. The system will provide information about the dish then.

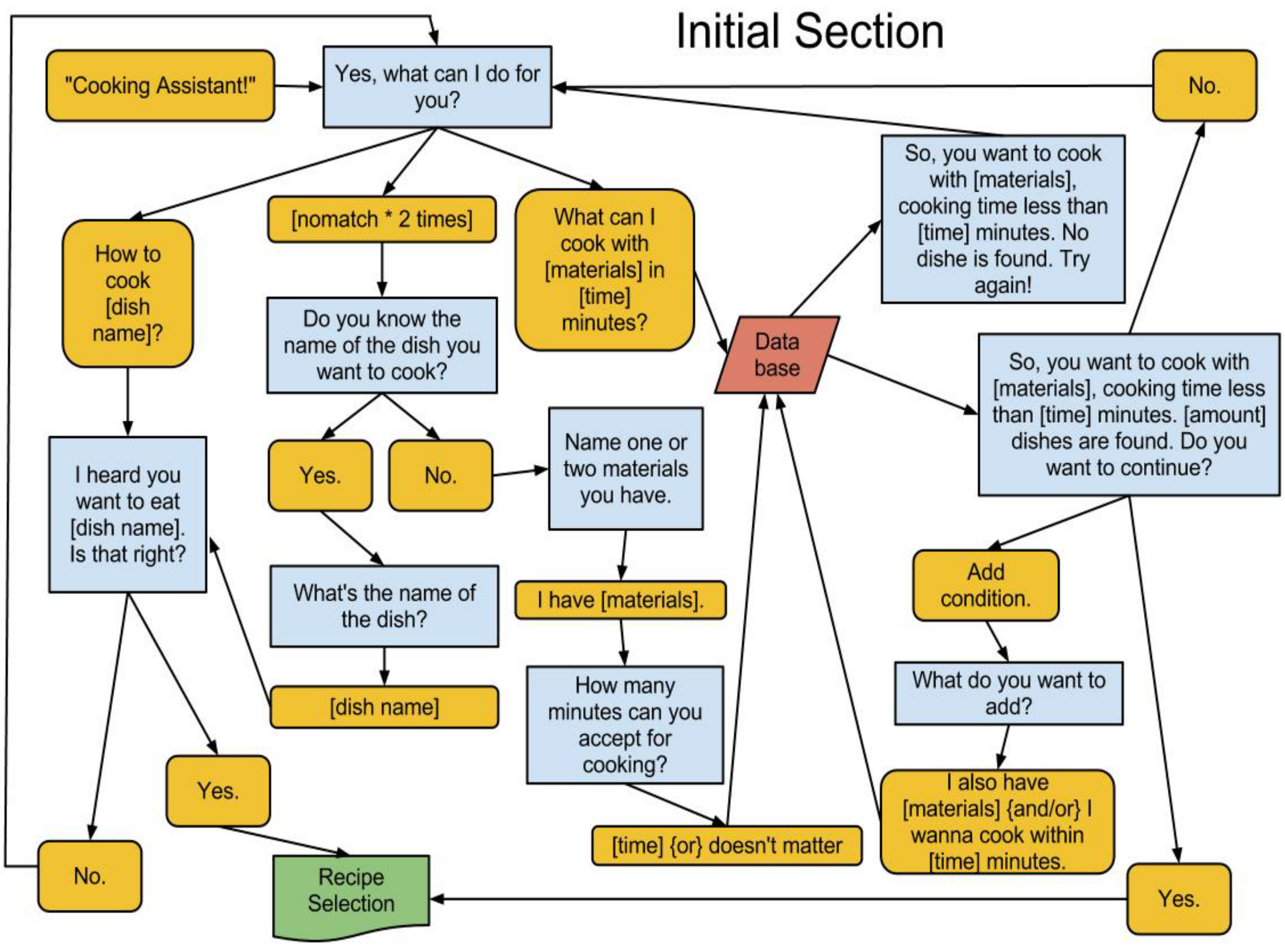
If there is more than one result, the system will list all the options with 3 options in each page. Then the user could select the dish by mentioning the dish name or browsing between pages by mentioning the page number or ask for the information about the dishes in the same way as described above. After the user chooses a dish, the followed processes are the same.

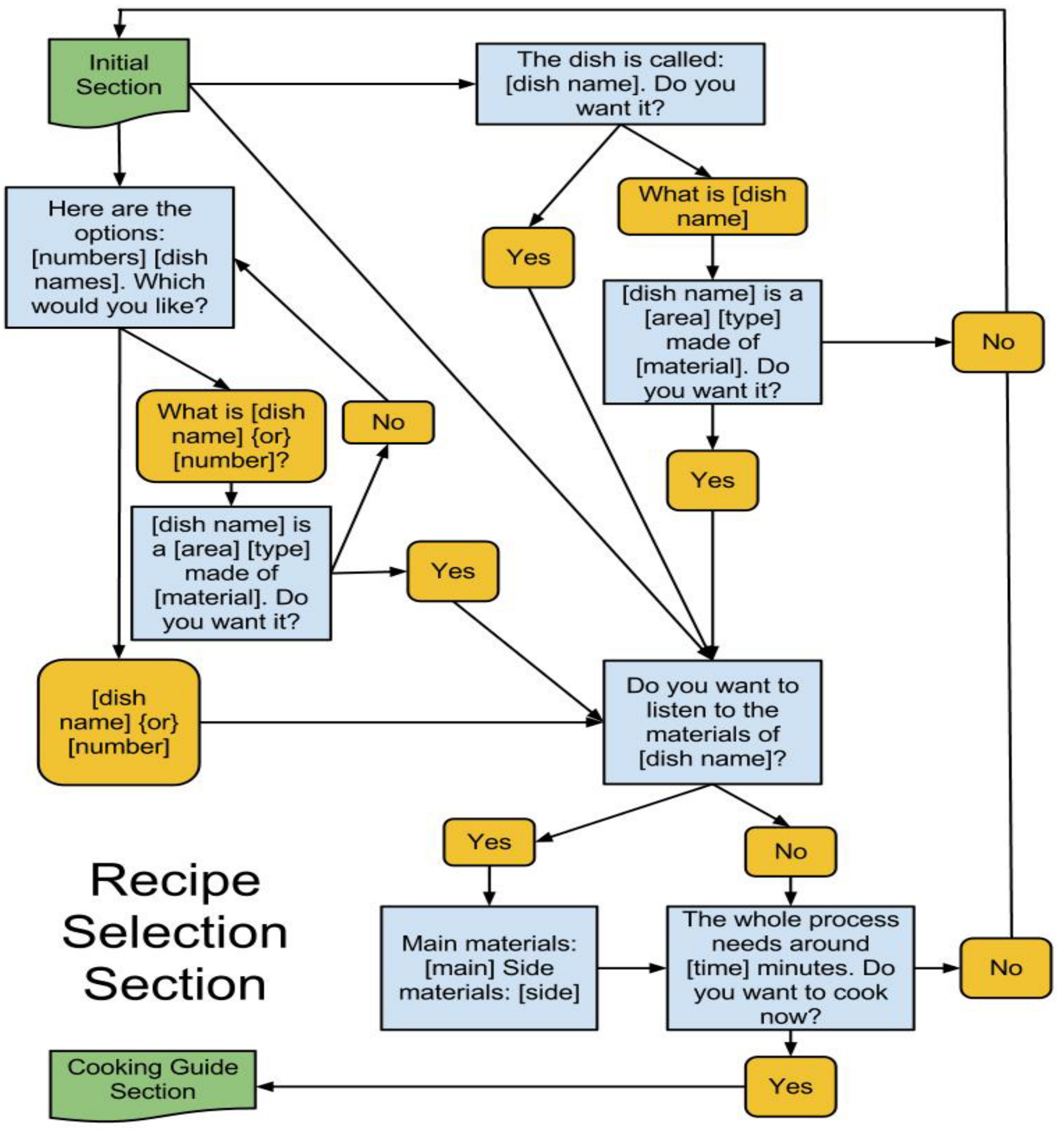
5.1.3 Cooking Guide Section

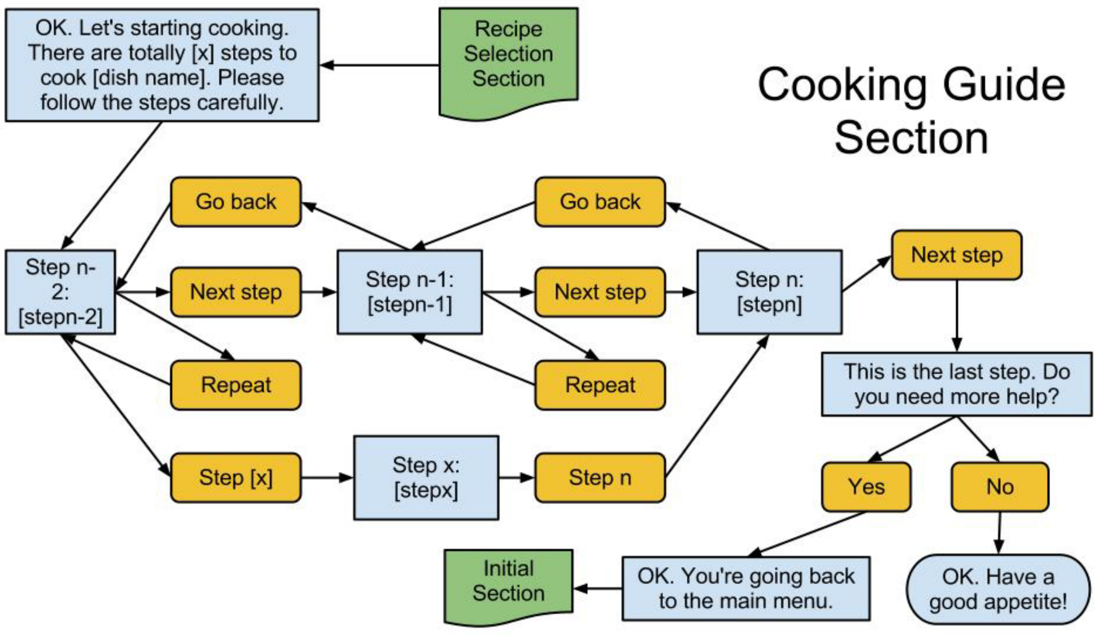
In this section, the system will starts with reading the first step in the recipe, and then wait for the user’s respond. The system will do nothing until the user says “next step”. Of course, during the whole cooking guide process, besides saying “next step”, the user could say “go back” to listen to the previous step, say “repeat” to listen to the current step again, or say “step n” (n means the number of any step) to jump directly to the n-th step. After reading the final step, the system will tell the user that the whole cooking step is finished and ask whether the user wants to go back to the initial section in order to cooking something else. If the user says no, the system will say “have a good appetite” and then goes back to the sleeping mode.

5.2 Flowchart

The flowcharts of all three sections are showed here respectively. The yellow rounded rectangles represent the users' inputs and the blue rectangles represent the system's discourses. The red parallelogram means accessing to the database. The green boxes stand for the entire sections and the only blue rounded rectangle is the end of the dialog flow. The arrows show the directions of the dialog flow.


Initial Section
"Cooking Assistant!"
Yes, what can I do for you?
No.
So, you want to cook with [materials], cooking time less than [time] minutes. No dishe is found. Try again!
How to cook [dish name]?
[nomatch * 2 times]
What can I cook with [materials] in [time] minutes?
Do you know the name of the dish you want to cook?
Data base
So, you want to cook with [materials], cooking time less than [time] minutes. [amount] dishes are found. Do you want to continue?
I heard you want to eat [dish name]. Is that right?
Yes.
No.
Name one or two materials you have.
Add condition.
What's the name of the dish?
I have [materials].
What do you want to add?
[dish name]
How many minutes can you accept for cooking?
Yes.
I also have [materials] {and/or} I wanna cook within [time] minutes.
[time] {or} doesn't matter
No.
Recipe Selection
Yes.


Initial Section
The dish is called: [dish name]. Do you want it?
Here are the options: [numbers] [dish names]. Which would you like?
Yes
What is [dish name]
[dish name] is a [area] [type] made of [material]. Do you want it?
No
What is [dish name] {or} [number]?
No
[dish name] is a [area] [type] made of [material]. Do you want it?
Yes
Yes
[dish name] {or} [number]
Do you want to listen to the materials of [dish name]?
Yes
No
Recipe Selection Section
Main materials: [main] Side materials: [side]
The whole process needs around [time] minutes. Do you want to cook now?
No
Cooking Guide Section
Yes


OK. Let's starting cooking. There are totally [x] steps to cook [dish name]. Please follow the steps carefully.
Recipe Selection Section
Cooking Guide Section
Go back
Go back
Next step
Step n-2: [stepn-2]
Next step
Step n-1: [stepn-1]
Next step
Step n: [stepn]
Repeat
Repeat
This is the last step. Do you need more help?
Step [x]
Step x: [stepx]
Step n
Yes
No
Initial Section
OK. You're going back to the main menu.
OK. Have a good appetite!

5.3 Files

Since the Voxeo platform supports only XML, VXML, and other static files rather than database and other dynamic files, most files need to be stored in the server side in order to retrieve information from the recipe database.

*.vxml is the extension name of VoiceXML files and they are static files which can be stored on the Voxeo platform.

The files ending with *.py are Python files which will be accessed through CGI. In the beginning of each *.py file, a variable called “cgipath” is defined, such as

```
cgipath = ‘http://192.168.1.1/cgi-bin’
```

which is used to store the path of the CGI files since the address of the server is always changing, therefore it is unnecessary to modify all of them in each file every time when testing the application.

The database file ends with *.db and is created according to SQLite database standard by SQLite Manager.

5.3.1 Voxeo Side:

Wakeup.vxml

This simple file is just used as a trigger for the application to wake up and go on to the initial section.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml xmlns="http://www.w3.org/2001/vxml" version="2.0">
    <noinput> </noinput>
    <nomatch> </nomatch>
    <form id="wakeup">
        <field>
            <option>wake up</option>
            <option>cooking assistant</option>
            <filled>
                <prompt>Hello! Welcome to use the system!</prompt>
                <submit next="http://Server_IP/cgi-bin/command.py"/>
            </filled>
        </field>
    </form>
</vxml>
```

5.3.2 Server Side:

5.3.2.1 Command.py

Command.py is the main file of the initial part which controls the main dialog flow and contains the grammar for taking the user’s command and search conditions.

The reason why to use a python file rather than a VoiceXML file here is that the grammar of the “searching by ingredients” part is very large since we never know what the users will say when they mention the ingredients that they have. The system should be very considerate that it must be able to recognize all the possible cooking materials, as well as the plural forms of the words. If we put all the possible words into the grammar included in a vxml file, the file will be too long for the system to access. Moreover, since the recipe database will be updated frequently and new materials will appear, it is impossible to cover them in a static VoiceXML file. Thus, to generate the grammar by a python program is easier and efficient; the program fetches the names from the database and make a copy of them with an “s” in the end as the plural forms, and store them into a variable in order to use it repeatedly instead of fetching them from the database again which causes so much burden to the database. In addition, since the <grammar> element supports neither CGI nor a variable, both the way of using a dynamic grammar (<grammar src=”*.cgi”>), or the way of using <grammar> to read a variable which is assigned by using <data> element to retrieve a CGI based grammar, do not work. After all, this command.py file generates both the grammar part and other VoiceXML functions.

As mentioned earlier, the system offers a function by which the user can add ingredients or other conditions such as cooking time if there are too many search results (for example, more than 10). In order not to create so many condition variables, such as condition1, condition2, condition3, condition4, etc. the following code can be used:

```
<assign name = ”condition” expr = ”condition + ’,’ + new_condition” />
```

Therefore, the variable will be re-assigned as the user adds new conditions. Even though the user adds iterative conditions, it doesn’t matter because in the python file, the condition variable will be split by ‘,’ and be trimmed into a set of strings (without iteration) in the trimcond.py which will be introduced below. (cf. 5.3.2.3 Trimcond.py) Therefore the system could repeat all the condition in a well-formed sentence and the users can understand easily.

5.3.2.2 Checkamount.py

This file takes the conditions requested by the user and conducts a search in the database. Then, it returns only the count of the results to the ongoing command.py file, which helps the user to decide whether to add a search condition or not.

The search conditions are introduced in a string separated by commas. Then they are transferred into a list and afterwards into a set in which, all repetitive conditions are reduced to one; therefore, no unnecessary access to the database will be conducted.

When conducting the search, the following SQL string is used:

```
'SELECT item FROM Recipestbl WHERE (main_material like "%s%"
or side_material like "%s%")
```

Firstly, only “item” (the name of the dish) is retrieved here because the other information is not needed, and this reduces the burden to the database. Second, “like” rather than “=” is used in order to make it case-insensitive, because the recipes in the database could come from anywhere or any culture and could be nonstandard.

5.3.2.3 Trimcond.py

This file collects all the search conditions from the user, reduces the repetition, and composes a sentence as a feedback giving to the user, for example, “you want to cook with tomato, pasta, and cheese, cooking time no more than 3 minutes.”

5.3.2.4 Recipeselection.py

Recipeselection.py is the only file of the recipe selection section. It is divided into several parts using “if…elif…” function, because it is then able to deal with several situations: searching by ingredients, searching by dish name, and searching by item ID (final retrieval after the user has a decision). So the programming code turns out to be more clear and organized.

```
if method == "ingredients":
# omission of several codes here
        sqlstr='SELECT * FROM Recipestbl WHERE ('
# omission of several codes here
        cur.execute(sqlstr)
        recipes = cur.fetchall()
        byIngredients(recipes)
elif method == "name":
        sqlstr='SELECT * FROM Recipestbl WHERE name like "%s"' % (cond1)
        cur.execute(sqlstr)
        recipe = list(cur.fetchone())
        byName(recipe)
elif method == "item":
        sqlstr='SELECT * FROM Recipestbl WHERE item = "%s"' % (cond1)
        cur.execute(sqlstr)
        recipe = cur.fetchone()
        byName(recipe)
```

There are three different conditions distinguished by if…elif…else function in the “searching by ingredients part”: only 1 result, 2 or 3 results, more than 3 results.

```
def byIngredients(recipes):
        if len(recipes)== 1:
  # omission of several codes here
        elif len(recipes)<=3 and len(recipes)>1:
  # omission of several codes here
        elif len(recipes)>3:
  # omission of several codes here
```

For the system to be considerate, listing three names at one time is acceptable, but it could be difficult for a user to remember more than three options in a voice-based situation. Not as vision which can be nonlinearly remembered by the users, voice is linear. Giving too much information at one turn would cause communication failure. Thus, when dealing with more than 3 results, they are divided into pages with 3 items in each and then the user can switch among pages by saying “next page”, “previous page”, and the page number directly, and choose recipes in a page. However, the users are suggested to refine the search conditions so that the searching results will not be so many. (cf. 错误！未找到引用源。)

5.3.2.5 Cookingstep.py

Cookingstep.py is the only file for the cooking guide section. The system introduces how many steps are there totally in order for the user to get a general idea.

```
<form id="step0">
        <block>
                <prompt>
                        There are totally %s steps to cook %s. Please follow the
steps carefully. <break/>
                </prompt>
                <goto next="#step1"/>
        </block>
</form>
```

If the user says “go back” at the first step or if the user says “next step” at the final step, the system will give friendly respond telling the user that it is already the first/last step, rather than give no feedback which will cause confusion to the user.

5.3.2.6 Recipes.db

This is the database file of the whole application. It contains all the recipes’ information and all the materials’ information which can be used for both the “searching by materials” and “searching by dish name” function. More description of the database will be given in the following section.

5.4 Database

There are 3 tables in the database: “Dish_namestbl”, “Ingredientstbl”, and “Recipestbl”. They are used to store the dish names, the materials, and the recipes respectively.

Even though all the dish names and the material names can be retrieved in the recipe table, two new tables are created in which all the dish names and material names mentioned in the recipe table are stored. They reason is that they can be retrieved more easily, without causing too much burden to the database and too much processing time, because in the recipe table, several names were written in one cell, which calls for more time to separate them into a list in Python.

5.4.1 Recipe Table

The recipe table is one of the core parts in this application which stores all the information of all recipes. Besides the index column, there are 7 columns in this table: name, time, type, area, main, side, and step. The name column stores all the dish names and the time column contains the approximate cooking time of each dish. The category information, such as main course, dessert, and soup, to which the dish belongs can be found in the type column and the information about where the dish comes from is in the area column. The main and the side column, of course, keep the information about the needed main materials and side materials respectively and materials existing in one cell are separated by semicolons. All the cooking steps are kept in the step column and separated by semicolons.

The reason for adding a time column is that the system should provide a piece of information before leading the user to the cooking steps: “The whole process needs around * minutes”. Therefore, the system can be more considerate, because the user can have a general idea of the cooking time and if it is too long for the user, the user can stop timely.

An idea of how to use the columns in order to make our system seem more intelligent is that: since the distinction of main and side materials still exists in the database, during the recipe choosing process, when the user faces several options retrieved from the database and asks “what is Swedish meatball” or “what is grilled salmon”, the system can give a feedback by combining the data from several columns, such as “Swedish meatballs is a Swedish [area], Main

Course [type], made by 2.5 pounds ground beef and onion [main]”. This idea makes the system more considerate for the user and it is also a good way to make full use of our database. Here is an example of one row in this table:

Table 3

ID	Name	Time	Type	Area	Main	Side	Step
1	swedish meatballs	50	main course	Swedish	2.5 pounds ground beef; onion;	1 egg; 1 cup milk; half cup dry bread crumbs; 2 teaspoons salt; half teaspoon black pepper; 2 tablespoons water; 1 and half cups milk; ……	Beat egg and add milk and bread crumbs; ……; Combine crumb mixture with cooked onion, salt, black pepper, ground beef, and ground pork; ……

5.4.2 Dish Name Table

Besides the index column, only one column exists in this table. All the dish names mentioned in the recipe table are copied to here. When the application starts, the python fetches all the names and put them into the grammar in VoiceXML, so that when the user mentions one of these names, it can be recognized.

Table 4

Row_ID	NAME
1	Swedish meatballs
2	Teriyaki Chicken
3	Layered Taco Casserole
4	Grilled Salmon
5	Fresh Peach Dessert
……	……

5.4.3 Material Table

Two columns are there in the table, besides the index column. The first column is used to store all the material names existing in both the main materials column and the side material column in the recipe table, since the distinction between main material and side material varies in cultures. (cf. 4.6 Vocabulary and Expressions) The second column stores the alternative names of the first column, for example, aubergine and eggplant. (cf. 4.6 Vocabulary and Expressions)

Table 5

Row_ID	MATERIALS	ALT_NAME
1	ground pork	
2	chicken	
3	onion	
4	milk	
5	bread crumb	
6	ginger	
7	Tomato	
8	potato	
9	chicken legs	
10	eggplant	aubergine
……	……	……

6. Future Development

1. One design direction of this application can be multimodal – voice and visual. Therefore, users can conduct all the process through both channels, and can also see the pictures from the screen. However, it should not disobey the original intention of designing this application as a voice-based one, because when using voice, people can have their eyes and hands free from the screen and then can concentrate on cooking.

2. It is an ideal situation that the user could say "what's next after the rice?" However, it is too ideal to program. It requires so many considerations in the grammar part in which all the possible expressions should be taken into consideration. However, in the current application, it will be better if the system tells the user how many steps totally there are before starting the cooking steps, so that they can have a general view of the following cooking steps, and it's easier for them to jump among the steps.

3. Another function could be added is the timer function which enables the user to set one or more timers when cooking. The timers can be named by the user arbitrarily and can also be cancelled.

4. As mentioned earlier, one problem of this system which caused so much inconveniences and troubles to the users and low down the system's usability is that the current low-developed TTS technology which makes the voice output hard to understand. Some solutions have been introduced above (cf. 4.5 Phonological Issues). However, in the future development, I think we can do it with the help of syntactic analysis, for example, we add comma after a verb if it is followed by a noun phrase, and add comma after the noun. Then, "add 3 gram salt into the mixture" will become "add, 3 gram salt, into the mixture".

5. It is better in the future to trim the data in the material column in the database. For example, "3 chopped tomatoes;" can be written as "tomato (3, chopped);" Therefore, data can be retrieved and used much more easily. For example, when the system introduces a dish, the names of the materials without the followed data in the parentheses will be retrieved; when the system reads the materials that the user need to prepare in order to cook a certain dish, all of the data are retrieved; when the user asks how many tomatoes are needed, we retrieve the first element in the parentheses.

6. The feedbacks (confirmations) in the dialog should still be reduced, since it is too fussy, boring and disgusts user. However, the reason why many confirmations are set here is that due to the low speech recognition rate, it would be better be if the system confirms with the user before going to the next process. In the further development, the dialog design could be like this: let the system go to the next step first, repeat a little of what the user said, and then, if it is a wrong recognition, the user could go back by saying "no, no", "I didn't said…" This design is very good on a communicative perspective, but it is a little hard to implement in VoiceXML now.

Appendix

A. Bibliography

1. Allwood, J., (1999). Are there Swedish patterns of communication? In H. Tamura(Ed) *Cultural Acceptance of CSCW in Japan & Norid Countries,*

Kyoto Institute of Technology, 1999. p. 90-120. Retrieve also from http://sskkii.gu.se/jens/publications/docs076-100/087.pdf

2. Allwood, J., Cerrato, L., Jokinen, K., Navarretta, C. & Paggio, P. (2005) The MUMIN annotation scheme for feedback, turn management and sequencing. In Allwood, J., Dorriots, B. & Nicholson, S. (eds.) *Proc. of the 2nd Nordic Symposium on Multi-modal Communication.* Gothenburg, Sweden: 91-109

3. Allwood, J. & Ahlsén, E. (2009) Multimodal intercultural information and communication technology – a conceptual framework for designing and evaluating multimodal intercultural communicators. From SSKKII Interdisciplinary Center, University of Gothenburg.

4. Batusek, R. & Kopecek, I. (1999) User interfaces for visually impaired people. Proceeding of Fifth ERCIM workshop on User Interfaces for All, Dagstuhi,Germany,1999

5. Jokinen, K. (2004) Communicative competence and adaptation in a spoken dialogue system. In proceeding of: INTERSPEECH 2004 - ICSLP, 8th International Conference on Spoken Language Processing, Jeju Island, Korea, October 4-8, 2004

6. Ondáš, I. S. (2006) VoiceXML-based spoken language interactive system. In: Proc. 6th PhD Student Conference and Scientific and Technical Competition of Students of Faculty of Electrical Engineering and Informatics Technical University of Košice, Košice, 17.5.2006, s. 97-98. ISBN 80-8086-035-1.

7. Sadek, D. (1999) Design considerations on dialogue systems: from theory to technology – the case of Aritimis -. In IDS-99, 173-187

8. Language technology, retrieved from http://en.wikipedia.org/wiki/Language_technology on Apr 21st, 2012.

9. Speech technology, retrieved from http://en.wikipedia.org/wiki/Speech_technology on Apr 21st, 2012.

10. Human–computer interaction, retrieved from http://en.wikipedia.org/wiki/Human-Computer_Interaction on Apr 21st, 2012.

11. Dialog system, retrieved from http://en.wikipedia.org/wiki/Dialog_system on Apr 22nd, 2012

12. Voice Extensible Markup Language (VoiceXML) Version 2.0, retrieved from http://www.w3.org/TR/voicexml20/ on Apr 21st, 2012

13. VoiceXML, retrieved from http://en.wikipedia.org/wiki/VoiceXML on Apr 22nd, 2012

14. Voxeo: www.voxeo.com

15. Voxeo, retrieved from http://en.wikipedia.org/wiki/Voxeo on Apr 22nd, 2012

16. Common gateway interface, retrieved from http://en.wikipedia.org/wiki/Common_Gateway_Interface on Apr 22nd, 2012

17. Python (programming language), retrieved from http://en.wikipedia.org/wiki/Python_(programming_language) on Apr 22nd, 2012