Language Generation and Speech Synthesis in Dialogues for Language Learning

by

Julia Zhang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Engineering in Computer Science and Electrical Engineering
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2004

Author . . . .
Department of Electrical Engineering and Computer Science
May 20, 2004

Certified by . . . .
Stephanie Seneff
Principal Research Scientist
Thesis Supervisor

Certified by . . . .
Chao Wang
Research Scientist
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Language Generation and Speech Synthesis in Dialogues for Language Learning

by

Julia Zhang

Submitted to the Department of Electrical Engineering and Computer Science
on May 20, 2004, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Computer Science and Electrical Engineering

Abstract

Since 1989, the Spoken Language Systems group has developed an array of applications that allow users to interact with computers using natural spoken language. A recent project of interest is to develop an interactive conversational system to assist students in mastering a foreign language. The Spoken Language Learning System (SLLS), the first such system developed in SLS, has many impressive capabilities and shows great potential to be used as a model for language learning. This thesis further develops and expands on SLLS towards the goal of a more sophisticated conversational system. We make extensive use of Genesis, a language generation tool, to complete a variety of natural language generation and translation tasks. We aim to generate natural, well-formed, grammatically correct sentences and produce high quality synthesized waveforms for language students to emulate. We hope to develop a system that will engage the user in a natural and realistic way, and our goal is to mimic human-to-human conversant interactions as closely as possible.

Thesis Supervisor: Stephanie Seneff
Title: Principal Research Scientist

Thesis Supervisor: Chao Wang
Title: Research Scientist

Acknowledgments

First, I would like to extend my most sincere gratitude to Stephanie Seneff, for giving me the opportunity to join the SLS group and work with her on this project. Her deep insight, suggestion, and critique have been invaluable guidance to me in the past year. Stephanie is an incredibly patient and understanding mentor, and it's been a great joy to work with her. The energy, enthusiasm and love she displays for her work is truly inspiring.

I'm very fortunate to have Chao Wang as my second advisor on this project. Chao is extremely knowledgable and has shared much of the burden in solving difficult problems of the system. I am very grateful to her for all the help and time she has given me.

I would also like to thank Scott Cyphers for being such a good sport, I can always count on him to help with debugging problems, big and small.

I want to thank my wonderful friends back home and at MIT. Thank you for the good conversations, the fun times and the bad, and the great memories.

Finally, I'm most indebted to my parents, for their hard work and the sacrifices they have made. They have given me love and unwavering support in both my personal and academic life, and have always encouraged me to find the fine balance between work and play.

Contents

List of Figures

Chapter 1

Introduction

Since 1989, the Spoken Language Systems group (SLS) in the MIT Computer Science and Artificial Intelligence Laboratory has developed an array of applications that allow users to interact with computers using natural spoken language. A sophisticated interactive computer system can potentially replace a large variety of services that are currently provided by humans, and is particularly useful in situations where one wishes to retrieve information from a large database. Many customer services, such as credit card companies or airline flight inquiry lines, already use conversational technologies in automated services to answer simple questions for the caller and defer to a human assistant when a question becomes too difficult to communicate between caller and computer.

Over the years, SLS has developed a wide variety of spoken language applications including weather information, airline flight planning/status, city guide and urban navigation, and restaurant search [8]. A recent project of interest is to develop an interactive system to assist students in learning a foreign language - a system that can help the user to improve their speaking and listening skills by engaging in a natural and meaningful conversation with the user. The Spoken Language Learning System (SLLS), is the first such system developed by an SLS student [6]. It enables users to engage in simple conversations with the computer system around popular topics such as family, occupation and personal information. The initial version of SLLS has many impressive capabilities and is, we argue, a good model for language learning.

However, many aspects of the system are worth exploring and improving upon. The focus of this thesis is to further develop and expand on SLLS towards the goal of a more sophisticated conversational system.

1.1 Motivation

Many people learn a foreign language for different reasons. For some, it is a necessity - business partners work and conduct meetings with their counterparts in foreign countries across the globe, scientists and researchers from all over the world benefit from exchanging information and discussing ideas on new discoveries and technologies. For others, it may be to feed a cultural interest, for ease of traveling, or to make new friends. Whatever the reason, to become fluent in a foreign language is an extremely difficult task, and requires much time and perseverance from the learner.

The main components of language learning consist of writing, reading, listening and speaking. In most cases, students have mastery over their writing and reading skills because these are emphasized in classrooms and can be acquired through efforts and practice on their own time. However, in order to effectively improve conversational skills, one must practice conversing with a person who is fluent in the language. Unfortunately, this is a harder task due to the lack of such a language partner, or simply because the student is not confident enough in his/her speaking skills to engage in conversation.

For years, SLS has been actively engaging in research projects involving speech recognition, natural language understanding, dialogue modeling, language generation and speech synthesis. Therefore, it is natural that the SLS group would hope to utilize its technologies to improve the language learning experience. The need for an effective language learning system puts us in an excellent position to utilize currently available technologies to launch the next version of a computer-human conversational system, and to advance the core speech and language technologies to meet new challenges.

Currently, SLLS supports conversation regarding exchange of personal information, family relations, hobbies, etc. The conversation flow has limited variability and follows a rather strictly-ordered script in the lesson plans configured by the teacher. It is limited in the number of ways the system can speak and recognize sentences. However, unlike an application in which the main purpose is to communicate some information, it is important for the language generation component of a language learning system to be reasonably complex. The dialogue spoken by the system should not only be grammatically correct, but also needs to be natural, well-formed sentences appropriate for language students to learn from. We hope to develop a system that will engage the user in a natural and realistic way, and our goal is to mimic human-to-human conversant interactions as closely as possible.

1.2 Goals

The focus of this thesis is to further develop the conversational capabilities of SLLS. We would like to increase flexibility and variability in both dialogue flow and sentence generation. We also aim to provide high quality speech synthesis for the system. In this version, the system is designed for Chinese speakers learning English. We will initially focus on conversations in the hotel domain - simulated dialogues that revolve around finding a hotel in the area, booking a room, checking in and other related questions and requests.

More specifically, the system will have:

- The ability to randomly generate a simulated hotel with simulated available rooms of different types and different prices,
- The ability to simulate a two-party conversation involving questions and answers about the rooms that are available in the simulated hotel in both English and Chinese,
- The ability to translate from English to Chinese for a set of sentences in the hotel domain,
- The ability to generate high quality synthesis in English for both sides of the conversation,

- The ability to parse and understand a corpus of utterances in English and Chinese appropriate for the domain.

1.3 Outline

The rest of this thesis is organized as follows. Chapter 2 introduces the background information on the core components of the SLS technologies on which this language learning system is based. Next, we give an overview of SLLS, the initial version of a language learning system, and highlight the areas where improvement is needed. Finally, we discusses the previous work done on language generation, and the current approaches to generation. Chapter 3 illustrates the overall architecture of the language learning system for the hotel domain and outlines the new additions to the system. Chapter 4 gives an overview of Genesis [2], the natural language generation system - followed by Chapter 5, where we demonstrate how Genesis is used for language generation and translation tasks in the system. In Chapter 6, we present an evaluation of our work. Finally, Chapter 7 summarizes our work and offers possible future research and expansions.

Chapter 2

Background

In the first section of this chapter, we outline the core components of SLS group's technologies, which were used as the basic building blocks of our language learning system. Next, we introduce SLLS, the initial version of a language learning system in the SLS group. In the last section, we summarize the current research landscape in natural language generation (NLG). We discuss the different approaches to NLG, and present the views of leading researchers on the advantages and disadvantages of each approach. Finally, we place Genesis, the language generation tool used in this thesis, in the context of this field.

2.1 Spoken Language System Group's Technologies

The development of a language learning system is an excellent research project for the SLS group because there exists a large pool of resources for us to draw upon. We are able to utilize several existing systems in the group to provide the basis for language learning. For some of these systems, we were able to integrate them into the language learning architecture with little or no modification, while others required some revisions to fit the needs of the system. The following is a description of each of the components.

2.1.1 Galaxy Architecture

Galaxy is an architecture for integrating speech and language technologies to create conversational systems. It adopts a client/sever architecture which allows users to communicate from light-weight clients to sophisticated servers that handle more computationally intensive tasks such as speech recognition, language understanding, database access and speech synthesis.

The Galaxy system makes use of a central hub to provide communication among components (i.e. servers) via a scripting language, which consists of a sequence of rules to instruct each component what command to execute and in what order to execute them. As shown in Figure 2-1, the typical servers involved in each turn [1] of conversation are: the audio/GUI servers, speech recognition, language understanding, context resolution, application back-end, dialogue management, language generation and text-to-speech conversion. At each turn, the audio server is responsible for recording the users' utterances and produces a digital waveform of the speech. This is streamed to the speech recognition component to output the most likely sentence strings. Next, the language-understanding server extracts the meaning from the sentence and produces a semantic frame representation. The context resolution server keeps track of the dialogue history, interprets the sentence in the context and resolves ambiguities. When it is appropriate, the application back end will retrieve the necessary information requested by the user from a database, ie. the result of a restaurant search or the status of specific flight information. On the basis of this information, the dialogue manager will decide what the system will now say in response to the user's inquiry and construct a meaning representation in the form of a semantic frame. The language generation component receives the semantic frame and is responsible for generating the appropriate string. Finally, the text-to-speech generation server will convert this text into speech [4].

The Galaxy architecture enables developers to rapidly create conversational systems for a wide variety of spoken language applications. Each component of the system is domain-independent (with the exception of the application back-end) and

[1] Each turn is defined as a dialogue exchange between the user and the system.

language independent, which is convenient for developing multilingual conversational systems. Domain and language dependent information, such as acoustic models for the recognizer, grammars for parsing and generation, etc., are stored in external files.

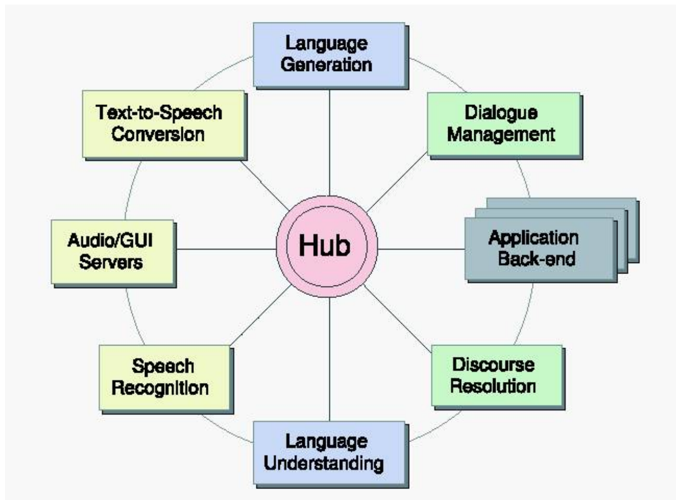


Figure 2-1: The Galaxy architecture

2.1.2 TINA

TINA [11] is a natural language understanding system which converts a sentence into a meaning representation. This language understanding system uses a probabilistic context-free grammar to parse sentences. Using a set of specified rules, TINA identifies the words of the sentences as grammatical components such as verb, predicate, clause. The parse tree is then converted into meaning representation of the sentence in the form of a semantic frame.

2.1.3 Genesis

Genesis is a language generation system which takes in a semantic frame representing the meaning of a sentence and produces the appropriate target string. It was first created in the early 1990s to serve as the generation component of the Galaxy architecture, but has since undergone two phases of significant improvements in design, looking to advance capabilities to generate natural language strings. The advanced system can produce strings in natural languages including English, Spanish, Japanese, and Chinese as well as formal languages such as HTML and SQL.

Genesis is a crucial tool in developing our language learning system. Genesis catalogs were created to produce both sides of the simulated conversations in both English and Chinese. When the user speaks an utterance or request, TINA parses the sentences into a semantic frame. This frame is used by Genesis to produce a (key : value) representation of the user's query and is the input to the dialogue manager. Genesis also played a big part in the task of translating English sentences into Chinese, which assisted in the process of developing a Chinese grammar for the sentences in the hotel domain.

2.1.4 Envoice

Envoice [12] is a concatenative speech synthesis system. The system selects and concatenates waveform segments from a pre-recorded speech corpus to produce natural sounding English, where concatenation can occur at the phrase, word, or sub-word level. The selection of a unit is determined by optimizing a cost function which is based on context and concatenation constraints. In this thesis, Envoice is used to provide high quality synthesis for the voice of the user in simulated dialogues.

2.2 SLLS

The Spoken Language Learning System (SLLS) is SLS's initial version of an online language learning system designed for English speakers learning Mandarin. The goal of the system is to improve student's speaking and listening skills by allowing the student to practice conversing with the computer system in the foreign language. In this section, we will describe how students, teachers, and administrators can use the system. Next, we will explain the way SLLS currently handles phrase management and dialogue management. Finally, we will highlight areas that can be improved in the second version of the language learning system.

2.2.1 SLLS Usage

Students, teachers, and administrators are all users of the system and each have a different role. What follows is a scenario of how a student can use the system, and what teachers and administrators can do to create lesson plans and maintain the website.

Student

Students who are interested in using the system to learn Mandarin can start by registering on the SLLS website. In this system, language learning is broken down into three stages: preparation, conversation, and review.

In the preparation stage, the student selects a lesson and studies new vocabulary and sentence phrases. Students can listen to the pronunciation of individual words or to whole sentences being spoken and follow along. Next, the student can listen to an entire simulated dialogue made up of phrases in the lesson to become more familiar with how the phrases can be used in conversation. This also gives the student an idea of what type of sentences the system expects and recognizes. When the student is confident with the lesson, he/she can move on to the conversation stage.

Lesson: relatives

Here is a sample of the phrases included in this lesson. You are not limited to these specific phrases, but what you say needs to follow these phrases structurally.

English	Pinyin	Chinese
hello	ni3_hao3. [**Listen**]	你好 [**Listen**]
goodbye	zai4_jian4. [**Listen**]	再見 [**Listen**]
how many sisters do you have	ni3 you3 ji3_ge5 jie3_mei4? [**Listen**]	你 有 姐妹 [**Listen**]
do you have any sisters	ni3 you3 jie3_mei4 ma5? [**Listen**]	你 有 姐妹 嗎 [**Listen**]
i have 3 sisters	wo3 you3 san1 ge5 jie3_mei4. [**Listen**]	我 有 三 個 姐妹 [**Listen**]

If you don't hear anything when you click on the links above, you need to get the all languages version of the **Java Runtime Environment** avaliable **here**.

You can also review a simulated conversation by clicking **here**. The simulated conversation will give you an idea of the typical flow of a conversation with the system.

During your conversation with the system, after you say something, the system will first try to paraphrase what you said as a confirmation, before generating a reply. If you are having problems trying to say something in Mandarin, feel free to speak in English. The system will attempt to translate what you said, and you can then repeat the Mandarin to progress the conversation.

When you are ready to, just click on where you would like SLLS to call you: **Home** | **Work** | **Cell**

Figure 2-2: Example practice phrases in SLLS to prepare the student for conversing with the system.

The student initiates a conversation with the computer system by clicking a button on the website to prompt the system to call a telephone number specified by the user. The conversation is expected to be similar to those of simulated dialogues, and the flow of the conversation (how the system will respond) is configured in a database for that lesson. During the conversation, if the student has trouble saying something in Mandarin, he/she can speak the sentence in English and the system will attempt to translate the utterance into Mandarin. The student can repeat the sentence in Mandarin and the conversation will continue. While the student and the computer are talking, the conversation is being transcribed. The website displays the system's responses and what the system believes the user is saying.

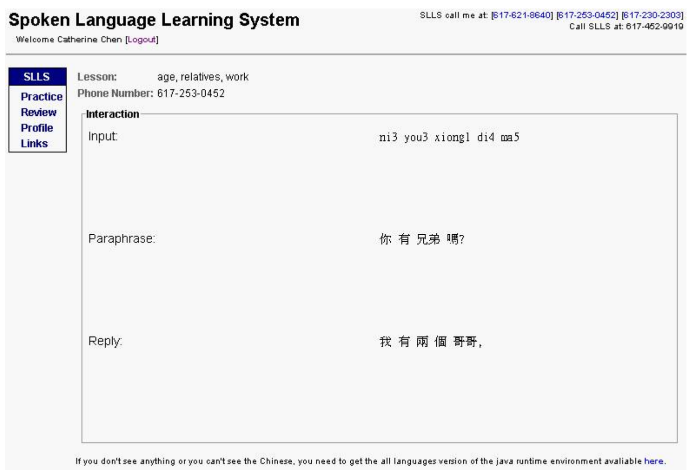


Figure 2-3: Visual feedback from SLLS during conversation.

When the conversation is over, the student can review his/her performance on the web page. He/she can listen to sentences that were spoken in conversation and get feedback on areas that need improvement. The words from the conversation transcription are color-coded to differentiate between words that were spoken well and those that scored poorly according to a confidence score, as shown in Figure 2-4.

Teacher

The teacher is responsible for creating lessons for the student's use. This includes deciding which new vocabulary and phrases will go into the lesson. The teacher has control over category management, phrase management, and dialogue management,

Figure 2-4: Review of performance from SLLS. Words in red had low confidence scores.

which are explained in more detail in later sections. The teacher can also keep track of a student's progress by reviewing their performance and leaving comments for guidance when appropriate.

Administrator

It is the responsibility of the administrator to maintain the website. He/she has control of user profile management and sets permission levels for different users. Administrators are responsible for fixing any reported bugs and updating the users on the status of the bugs. In addition, the administrator also shares with the teacher the tasks of category management, phrase management and dialogue management.

2.2.2 Category Management

First, we will introduce the concept of categories. A category contains an equivalent class of words that can be used interchangeably in the same sentence. SLLS uses categories to group sentences of the same type. For example, the following three sentences *"I have one brother"*, *"I have two sisters"* and *"I have three uncles"* can be represented as *"I have %COUNT %RELATION"*. A word preceeded by % is the name of a category. In this case, the words *"one"*, *"two"*, *"three"* are elements of the COUNT category and the words *"brother"*, *"sister"*, and *"uncle"* are words in the RELATION category. Users with the right permission can create and delete categories, as well as add and remove elements from a category.

2.2.3 Phrase Management

To add a new phrase into the lesson, the administrator needs to know two things. First, the key-value parse of the new phrase is needed. This parse is obtained by running the sentence through TINA, the natural language understanding system. The second thing the administrator needs to specify are the categories the new phrase will reference. The process of adding new phrases is quite tedious and we hope to improve this process in the second version of the language learning system.

2.2.4 Dialogue Management

The administrator can control the dialogue flow of the conversation by specifying how the system should respond to any particular utterance spoken by the user. First, the user's utterance is parsed by TINA into a key-value representation. This parse is used as a key to perform a lookup on the **SLLS_Dictionary** table, where a dictionary ID is returned to us. Using the dictionary ID and the lesson ID, we can do a lookup in the **SLLS_Dict_Reply** to obtain a reply. Both the teacher and the administrator can modify the **SLLS_Dict_Reply** table to specify phrases that can be used as replies to a user's utterance. If there are more than one reply corresponding to dictionary ID and lesson ID combination, then a reply is chosen at random.

For example, if the user asks "what do you do for a living", the key value representation for this phrase is: *clause: wh_question; pronoun: you; topic: profession.* The dictionary ID for this parse in the **SLLS_Dictionary** table is 20. Say the student is studying a lesson on Profession, which has a lesson ID of 3. Using this ID combination, we can retrieve a reply in the **SLLS_Dict_Reply** table and return the reply *"I am a %PROFESSION"*. A word is selected at random from the category PROFESSION, and a possible reply may be *"I am a doctor"*.

This table look up approach for dialogue management puts a strict limitation on the variability of the conversation flow. Each utterance can only be followed by the phrases that are in the reply table added by a teacher or administrator. As the size of the tables grows, maintenance also becomes difficult. Moreover, the issue of how the sentences are spoken is also constrained. For each sentence, the category tags (i.e. %COUNT) are the only places where we have variability.

In the next chapter, we will present modifications we have made to the modules in the system for the hotel domain to address these concerns.

2.3 Language Generation

Natural language generation capabilities are needed in a wide range of today's intelligent systems. There exist many different approaches to language generation for varying degrees of generation complexity. The different approaches to language generation can be categorized into two groups: template-based generation and linguistic generation. In this section, we introduce the two main categories and discuss the pros and cons of each.

2.3.1 Template-based Generation

Template-based generation is defined as one in which most of the generated string is static, but some of the words are dynamic and can be filled in with different word choices. An example of this is a program that displays a greeting message to the user: *"Welcome <name>!"*, where *<name>* is replaced by the name of the user.

Template-based systems tend to have very few linguistic capabilities but can handle simple tasks such as substitution of pronouns and verbs in a phrase and ensuring subject-verb agreement. Although template-based systems may appear to be primitive, many commercial systems and research systems, such as CoGenTex (Ithaca, NY) and Cognitive Systems Inc (New Havens, CT) [2] employ template-based generation components.

Why use Template-based Generation?

The most notable advantage of template-based systems is their simplicity. In most software programs where generation capability is needed, a template-based module is sufficient to handle the generation demands and needs. Linguistic generation, on the other hand, requires much more time and planning to write a comprehensive set of grammar rules. Furthermore, as Ehud Reiter notes, "there are very few people who can build NLG systems, compared to the millions of programmers who can build template systems" [10].

The linguistic generation approach requires that the system use an intermediate meaning representation for the information to be generated. In *NLG vs. Templates*, Reiter states that most systems do not use such a representation and the cost of implementing one is high and often unnecessary. Reiter provides the following example where template-based generation is more cost effective and appropriate for the generation needs [10]. Consider a software program that prints out the number of iterations performed in an algorithm. All possible output strings are in the form:

- *0 iterations were performed.*
- *1 iteration was performed.*
- *2 iterations were performed.*

In this case, template-based generation is sufficient to substitute the correct number of iterations and handle the subject-verb agreement. If the linguistic generation approach were used, developers would have to implement an additional component that would translate concepts such as *iteration* and *algorithm* into a syntactic representation. This would be time consuming and wasteful for the task at hand.

2.3.2 Linguistic Generation

Linguistic generation is an elegant and more sophisticated approach for language generation. Generation is based on a comprehensive set of grammar rules of the targeted language specified by the system developer. The input to a linguistic generation model contains a great deal of linguistic detail including those used for syntactic structure and features.

Why use Linguistic Generation?

There are clear advantages to the linguistic approach to natural language generation. Even template-based approach advocates agree that linguistic approach generations tend to be more correct and of higher quality because they can take advantage of the large amount of linguistic information embedded in the meaning representation [9, 10].

Another notable advantage of linguistic approach generation is the ease of maintenance. Suppose the system developer decides to change the time constructs from a military format to an *"o'clock"* format. This change can be reflected in modifying one rule rather than making changes in every place that has a time format in the template-based system. In many cases, the time invested in developing rules is worthwhile for making the system more robust and manageable and cutting down maintenance time in the future.

2.3.3 Genesis

Genesis is a generation tool that follows a hybrid technique for language generation, with a linguistic component and a non-linguistic component. Developers of Genesis made this decision by considering the role of Genesis in the Galaxy system. Galaxy will hand to its generation component a meaning representation of varying degrees of complexity. On one extreme, Genesis will receive hierarchical, linguistic meaning representations, consisting of key-value pairs, lists, clauses, predicates and topics. It can also receive simple flattened "e-form" (electronic form) representations, made up of only simple key-value pairs and lists.

Essentially, Genesis developers wanted to create a framework that allows domain experts to quickly and efficiently develop knowledge bases for simple, as well as complicated domains. We believe that their goal has been accomplished. As we will see in later chapters, we used Genesis for both simple template-based generation and challenging linguistic-based generation to complete the tasks in this thesis.

Chapter 3

SLLS for Hotel Domain Development

The Spoken Language Learning System for the hotel domain is an expansion of SLLS. It adapts the web-based language learning format of SLLS and supports the hotel domain. Several components of the system are new in this second version of SLLS. Some pieces - such as the language generation module, which we discuss in Chapter 4 and 5 - are modifications to the original SLLS intended to improve conversation capabilities. Additionally, there are several pieces that were implemented specifically for the development of the hotel domain system.

3.1 Random Hotel Generator

Currently, our language learning system will focus on supporting user/system conversations in the hotel domain. In order for the conversations to avoid repetition and encompass a large degree of variety, we would like each dialogue to refer to a distinct hotel. The Random Hotel Generator was developed by a student in SLS to serve this purpose.

At the start of simulation for each new dialogue, the hotel generator produces a hotel with a random set of hotel attributes. These properties include the size of the hotel (i.e. number of floors, rooms per floor), availability of rooms, extra features of the rooms, room prices and other general hotel traits. The generation of these attributes is tailored using an XML configuration file. Administrators can use this configuration file to specify things such as the price range each hotel room should fall in, a list of features of the hotel such as a pool or business center, a percentage range for room occupancy, etc. During the entire conversation, the system will refer to this randomly generated hotel as the source of information when responding to users' inquiries about the hotel and finding a hotel room. In the future, if the system were to be upgraded and developed to an automated hotel search service, a database containing information of actual hotels can replace the random hotel generator.

3.2 User Simulator

When the student engages in conversation with the language learning system, he or she will play the role of the person inquiring about hotels, booking a hotel, and checking in; the system will act as the hotel representative and assist the user in his or her requests. Before this stage, we have developed a component that acts in the place of the actual user to simulate inquiries and requests so that we may provide two-sided simulated dialogues for the student to study and review. In addition, it is useful to have such a simulated user module to guide us in implementing and debugging the system. The system can be utilized in two distinct modes. In the first, we can manually dictate what the user will ask or request, with no regards to what the system previously responded. In the second mode, the simulated user will produce utterances depending on what the system had just said, in the form of simple meaning representation frames. With a working user simulator, we can then produce simulated dialogues the student can study and review in the preparation stage, before he or she attempts to converse with the system. Moreover, we can generate a large amount of simulated user sentences via batch mode processing and use these simulated utterances to train the language model for the recognizer.

3.3 Turn Manager

As we have described in the previous chapter, the original version of SLLS requires several stages of tedious manual configurations for the setup and maintenance of phrase management and dialogue management. In the language learning system for the hotel domain, we hope to make this process more transparent and enhance the system's conversational capabilities. We will eliminate the concept of word categories, as well as the need for phrase management, and replace the table look-up approach for dialogue management with a dialogue manager server. To formulate a system response for the user's utterance, we divide the task into two parts: *what* the system will say, and *how* the system will say it.

The Turn Manager is responsible for taking a e-form encoding the meaning of the user utterance as a set of key-value pairs, and producing what the system should respond in the form of a semantic frame. This frame is then passed to the language generation component to construct a natural response string. The language generator of the system is the core component of this thesis, and will be explained in great detail in the following two chapters. We believe this design of the Turn Manager will increase complexity in the current SLLS conversation flow, and engage the user in a manner that more closely resembles human-to-human dialogue interactions.

3.4 Envoice

Just as it is important for a language learning system to have a sophisticated language generation module that can produce grammatically correct, well-formed sentences for the student to learn from, it is equally important that the speech synthesis component be able to produce quality synthesized sentences for the student to follow along. In this thesis, we used Envoice - a concatenative speech synthesis system, to provide the voice of the simulated user. Our goal is to produce quality speech synthesis to be used for the voice of the simulated user.

Envoice selects and concatenates waveform segments from a pre-recorded speech corpus to produce natural sounding English, where concatenation can occur at the phrase, word, or sub-word level. However, since the corpus has good coverage of the words expected from the domain, the synthesizer chooses whole words or phrases as segments, as it naturally prefers larger units to optimize the cost function. In addition, Envoice provides begin-time and end-time information on each word for its synthesis. Using this feature, we are able to allow the user to listen not only to the whole sentences in the dialogues, but also to individual words in the phrase.

We start by recording a basic speech corpus in the hotel domain from which Envoice will select appropriate waveform segments and concatenate to form other system responses. In order to produce quality speech, the pre-recorded speech corpus must include each word embedded in all appropriate prosodic contexts, in order to yield the appropriate prosodic realization for every possible situation.

Next, a speech recognizer produces an initial word-to-phonetic alignment, which will then be checked and manually corrected. Each utterance recorded is aligned manually to make sure all the words are complete and crisp when the synthesizer chooses words from different recorded sentences to synthesize a new sentence.

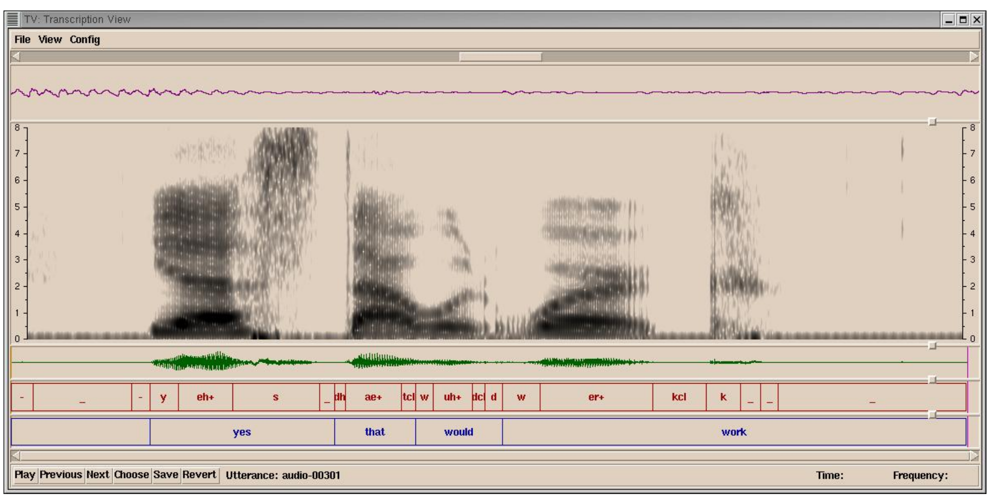


Figure 3-1: Interface for manual alignment of recorded waveforms.

We used the process of recording-alignment-evaluation to produce quality synthesis that covers the hotel domain. It is often the case that the word or words the synthesizer needs to form a new sentence are available in several places in the recorded speech corpus. The synthesizer does not always pick word units from the most ideal recorded utterance, and there exist choices in other places that would result in clearer or more complete synthesis. Envoice selects words using a unit selection algorithm that optimizes based on concatenation constraints. For this task, we do not explore concatenation constraints or attempt to modify the rules Envoice follows for word selection - as this is an area that is beyond the scope of this thesis. Instead, we turn to Genesis to provide a simple solution. For certain words or phrases that are identified to result in less than acceptable synthesis, we can specify a "shortcut" to a waveform in the Genesis catalogs so that the synthesizer will always pick the clearest form of audio output. We will explain this process in more detail in Chapter 5.

Currently, the voice of the system - in both simulated conversations and in actual conversation with a student user - is provided by Dectalk.

3.5 Simulated Dialogues

Each time students wish to study the hotel phrases in preparation for conversing with the system, we would like to present them with a new simulated dialogue. In the beginning of each new simulation, the Random Hotel Generator creates a new hotel with its own set of attributes regarding room prices and availability which the simulated conversation will reference. The dialogue is displayed in both English and Chinese, so that the students can conveniently reference the conversation in their native language. As previously mentioned, we run the user simulator in batch mode to provide the user side of the conversation. The system side is generated by a Turn Manager. The process is illustrated in Figure 3-2. Initially, a welcome frame is generated containing the first utterance, to launch a simulated conversation. This frame is passed to the language generation component to convert the frame into English and Chinese inquiry strings. Additionally, the frame is passed to the turn manager for a reply frame. Similarly, the language generator converts the reply frame into English and Chinese responses, and the reply frame prompts the batch mode simulator to produce the next request or question. This process continues until a hotel room that satisfies the user's request has been found.

This simulation run results in a log file that contains the entire conversation in English and Chinese strings. Next, we run a hub script to synthesize waveforms for each utterance in the conversation using two voices to differentiate between the simulated user and simulated system. In this step, a second log file is created such that for each utterance, the English and Chinese strings are available, as well as the synthesized waveform in English. Finally, a Java program extracts the necessary information to automatically generate an html file similar to the one in Figure 3-3. The student can listen to each English sentence being spoken, and in addition he or she can click on individual words in the sentence and listen to their pronunciation. The student can consult the Chinese translations if they have trouble understanding the English sentences.

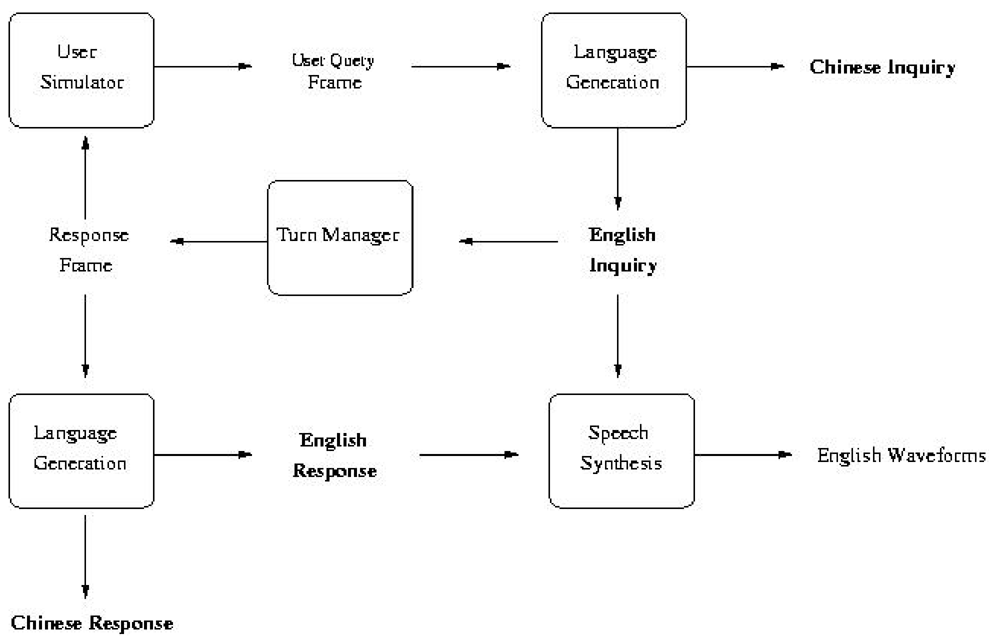


Figure 3-2: Flow diagram of simulated dialogue generation.

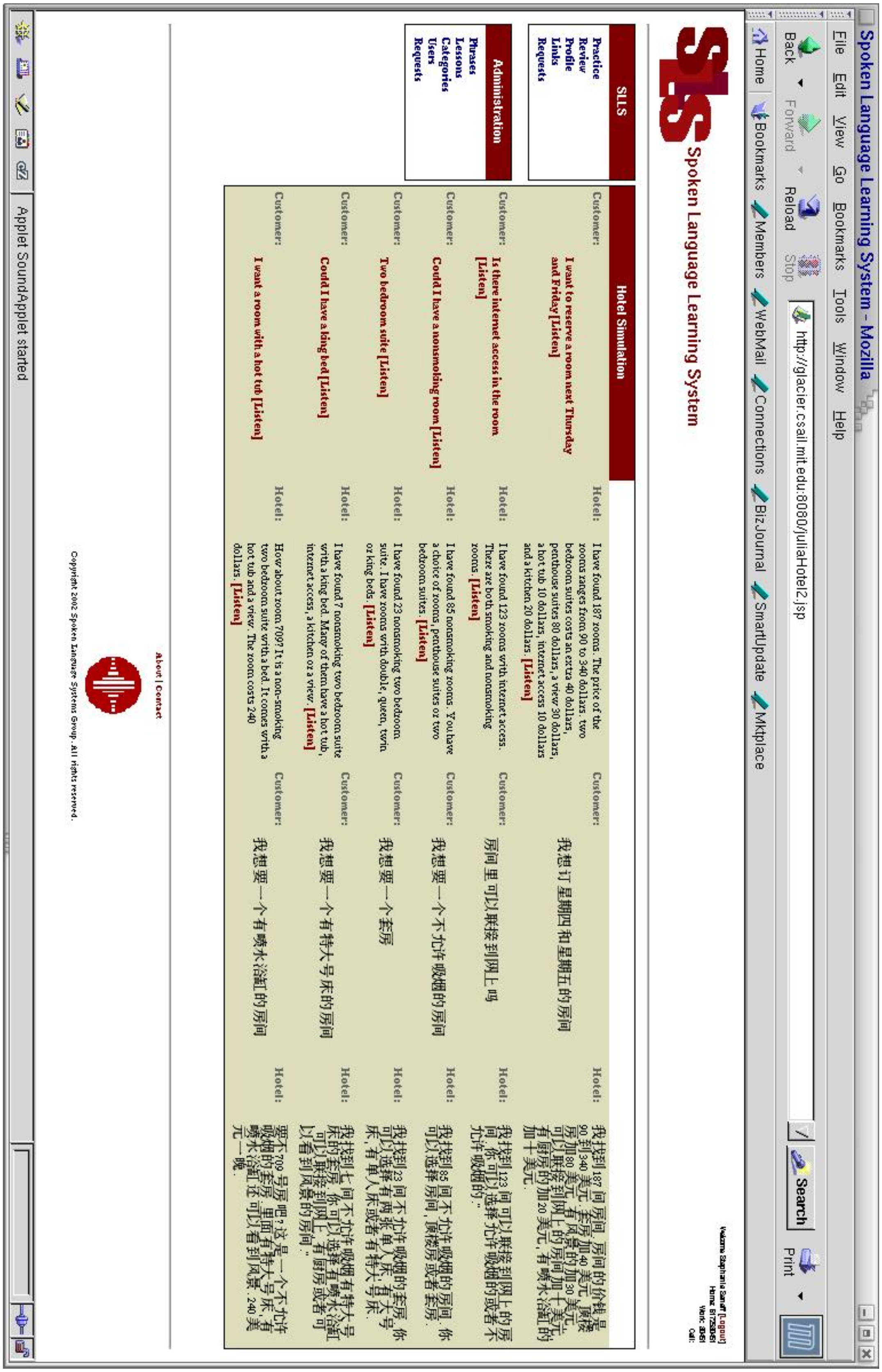


Figure 3-3: Example of a hotel simulated dialogue.

Chapter 4

Overview of Genesis

Genesis is a language generation tool for converting semantic frames into natural languages such as French, Chinese and English, or into formal language such has HTML and SQL. In this chapter, we will describe the components that are required for generation using Genesis, and explain several features to help us illustrate the capabilities of Genesis.

4.1 Semantic Frames

In the Galaxy system, we use semantic frames to represent the meaning of a sentence. Frames have hierarchical structures containing key-value pairs, where values can be in the form of a string, a number, a list, or another frame. This recursive structure of frames allows us to encode sentences of varying lengths and complexities. There are three different types of frames: *clause* (c), *predicate* (p) and *topic* (q). Clause frames represent sentences and complements. Predicate frames are primarily for prepositional, adjective, and verb phrase, and topic frames are mainly for noun phrases. Below is a simple clause frame that represents the sentence *"Do you have any apples?"*

```
{c yn_question
   :aux "do"
```

```
:topic {q pronoun
          :name "you"}
:pred {p possess
         :topic {q food
                   :quantifier "any"
                   :name "apple"
                   :number "pl"}}}
```

The frame above shows a clause frame titled *yn_question* (yes/no questions) that contains three key-value pairs. Notice that there are other frames nested in the *:topic* and *:pred* keys. The *pronoun* topic frame and the *possess* predicate frame are called the child frames of the *yn_question* clause frame.

In generating a string from a semantic frame, Genesis steps through the frame using a top-down, depth-first path. A frame can have access to information in its child frame without stepping into the child frame; however, there are no back pointers from children to parents. In order to resolve this issue, an additional "info-frame" is used. The info-frame is a place where parent frames can put context information for the child frame to access at a later point. Its purpose will be clearer once we introduce some Genesis commands and show some examples. For now, we can treat the info-frame as a global storage place where information can be accessed anywhere in the frame.

4.2 Linguistic Catalog

A linguistic catalog specifies the rules for generating strings for a particular domain and language. A catalog has four components - preprocessor, grammar, lexicon and rewrite rules - which are defined by the contents of the *.pre* file, *.mes* file, *.voc* file and *.rul* file, respectively. Developers can specify generation rules for a particular domain and language by creating these files. In the next several sections to follow, we describe the contents of each rule file and its role in generation.

4.2.1 Preprocessor

The preprocessor contains a set of rules that modify the semantic frame before passing it to the grammar component for generation. In this stage, we often wish to add additional key-value pairs to the frame that will give the grammar extra information and assist in the generation process. We can also run a command on parts of the frame. For example, if a key has a list of values for its attributes, we may wish to sort the list in alphabetical or numeric order so that the grammar component can list the values in order in its generation.

4.2.2 Grammar

The grammar is considered as the core of the catalog - it is the set of rules that will handle generation of any input semantic frame in the domain. We realize that even within a particular domain, there can be an infinite number of input frames to be handled. Genesis combats this problem by having template-based rules, as well as the capabilities to make recursive calls in the grammar rules.

A grammar rule consists of two parts, a rule name and a rule body. Consider the following topic frame:

```
{q name
   :firstname "Jane"
   :lastname "Doe"}
```

and the grammar rule:

name *:firstname :lastname*

This rule specifies that, to generate a frame titled *name*, we evaluate the value of the keys *:firstname* and *:lastname*. In this case, we get the vocabulary entries *"Jane"* and *"Doe"* and proceed to look in the lexicon for the default strings. If the value of *:firstname* or *:lastname* were another frame, then we would search for a rule in the grammar to handle the child frame. The rules in the grammar are usually much more complex, exploiting a rather rich set of commands which are listed in Appendix A.

Groups

Instead of having a separate rule for each frame, we can group together the frames that have the same processing requirements and have one group rule-template to process all the frames in that group. Grouping rule-templates have the following format:

<type_group>	*group-name*$_1$ *group-name*$_2$ ... *group-name*$_n$
<group-name$_1$*>*	*group-member-list*
<group-name$_1$*>_template*	*commands*
......	
group-name$_n$	*group-member-list*
group-name$_n$*_template*	*commands*

The frames within a group must be of the same type. The *<type>_group* rule template lists the groups for each type (*clause*, *topic*, *predicate*), and each *group_name* rule lists each member of the group by its frame name. Lastly, the *<group-name>_template* specifies the commands to process the frames in the group.

Default Rule Template

To make sure that we can handle generation for any incoming frame, we can have a default rule template for each of the three frame types: *clause*, *predicate*, and *topic*. Any frame that does not have its own rule, and does not belong in any group, will be handled by the default rule. Default rule-templates have the following format:

<frame-type>_template　　　*commands*

Rule Template Finding Algorithm

Genesis follows the following algorithm to find a rule with which to process an input frame:

1. If there is a rule-template that matches the **frame name** of the current frame, use that rule-template.

2. If there is a **group** that contains the frame as a member (i.e., the name of the frame is on its membership list), then use the group's rule-template.

3. Use the **default template** for the type of the current frame (i.e., clause, topic, or predicate).

4.2.3 Lexicon

The lexicon is a set of vocabulary items, where each item contains some linguistic information about a particular vocabulary word. The lexicon has the following general format:

entry-name *part-of-speech* *"default"* *other-lexical-information*

The entry-name is the vocabulary word that is being looked up. The *part-of-speech* is usually one of N (noun), V (verb), A (adjective), X (auxiliary), P (proper noun), or O (other). Every entry has at least the *part-of-speech* and the *default* string, but the amount of *other-lexical-information* varies from entry to entry. *other-lexical-information* can indicate other forms the entry can appear in, such as the plural form, or masculine/feminine form.

Consider:

bus *N* *"bus"* *:plural* *"buses"*

This denotes that the string "bus" has a default generation of "bus". However, a grammar rule can also explicitly specify to generate the entry in its plural form, in which case it will return *"buses"*.

Setting Tags and Adding Key-Value Pairs in Lexicon

In the lexicon entry, we can set up tags or append other key-value pairs to be used for generation at a later point. Consider the following lexicon items:

hourly_rate	*N*	*"hourly rate"*	*; $:money*
1	*O*	*"1"*	*$:money "one dollar"*

The rules above tell us that if we look up the lexicon *"hourly_rate"*, we will get the default generation *"hourly rate"*. In addition, we set a *$:money* tag, so that we know this phrase talks about money. In a subsequent part of the sentence, when we generate *"1"*, we will return *"one dollar"* rather than the default string *"1"*. Of course, this may not be what the developer wishes to do, but it is a convenient capability of the lexicon rule in Genesis. Similarly, rather than just setting a tag, we can also add another key-value pair that will be used later in the sentence.

Multiple Generations

There need not only be one default generation for a particular lexicon. We can allow a lexicon entry to have multiple generations and one will be picked at random when the grammar calls on this entry. We do this by setting the *part-of-speech* to *Cycle* in the following way:

i_want_a	*Cycle*	*"Can I have a"*
i_want_a	*Cycle*	*"I would like to have a"*
i_want_a	*Cycle*	*"I would like a"*

This denotes that to generate the lexicon entry *"i_want_a"*, one of the three default strings will be picked at random. This feature allows us to have a large degree of variability in generating our conversational dialogues. The same semantic frame can result in vastly different strings each time it is generated. This resembles natural human speech habits - since we tend to express the same meaning in different ways each time we speak them.

4.2.4 Rewrite Rules

Finally, the last component of the catalog is the rewrite rules, and is the final component to have an effect on the string in the generation process. The rewrite rules use a match-and-replace method to refine the preliminary target string generated thus far by the grammar and lexicon. Each rewrite rule consists of two strings in the following format:

"old string" *"new string"*

The left string specifies the pattern to be matched in the preliminary target string, and the right hand string specifies the new string that will replace it.

A typical application for rewrite rule might be to convert the indefinite article *"a"* to *"an"* before a word beginning with a vowel. For example, the grammar and lexicon may generate the string *"a idea"*. We can use the following rewrite rule to correct it to *"an idea"*.

"a idea" *"an idea"*

Other uses of the rewrite rule may be to correct spaces and punctuations. Developers are encouraged to use the grammar and lexicon to specify generation rules when possible and keep the number of rewrite rules to a minimal. The primitive pattern match feature should only be used to correct surface problems, such as the above example, to keep computational cost and time from being an issue.

Chapter 5

Language Generation for Dialogues Using Genesis

In this chapter, we will describe the different Genesis catalogs that were developed for this thesis, and explain the purpose and contents of each.

5.1 English Inquiry

English inquiries refer to the user's requests and questions concerning hotel room. Examples include phrases such as *"Do you have a room next Wednesday?"* and *"Does the hotel have a pool?"* English inquiries are generated from the inquiry frames produced by the user simulator, and are used in simulated dialogues.

The inquiry frames are single-level frames without linguistic information, and consist of only key-value pairs. This makes the grammar rules in the catalog rather straightforward, as it does not need to be concerned with extracting subject-verb agreement and tense information from the frames. Moreover, since the frames are not multi-leveled, we do not need to use commands such as *tug* and *yank* to access information for generation from the parent frame in the child frame. An example of a batch mode frame is shown below, and the English string generated from it may be *"I want to reserve a room Monday and Tuesday"*.

```
{c user_response
   :weekday "monday"
   :ndays 2"}
```

In the lexicon file of the English inquiry catalog, we utilize the *Cycle* function to generate multiple English sentences from one meaning representation. For example, the frame above can generate *"I'd like to check in on Monday for two days"*, *"I'd like a room Monday and Tuesday"*, *"Do you have a room available Monday for two days"*, and more. This eliminates monotony in the simulated dialogues and at the same time, instructs the student of several ways to express the same meaning in English.

5.2 English Responses

English responses are the system's search results to the user's request with a set of constraints, and answers to the user's questions. English responses are generated from meaning representation delivered by the Turn Manager, and their structures are relatively more complicated than the inquiry frames. Most reply frames have a hierarchical structure containing key-value pairs where values can be in the form of a string, number, list or another frame. These reply frames are titled according to the kind of response they provide, such as narrowing down the room search and proposing a room, which will match different rule templates during generation. The complexity of the frames allows us to explore advanced features in Genesis for language generation.

5.2.1 Example 1: Preprocessor

Below is a *singled_out_category* frame that is returned by the Turn Manager after the available rooms have been narrowed down using the user's most recent constraints. At this point, there are still too many available rooms to describe them individually in a natural manner. Instead, the Turn Manager prompts the user to give another constraint by speaking about one aspect of the room - in this case, about the room prices. In the English string to be generated from this frame, we would like to include the range of prices for these rooms, and explain the pricing policy for different features in the room. This will give some guidance to the user as to how to give the next constraint.

```
{c singled_out_category
   :speak_about {c price
                   :pricing_policy {c pricing_policy
                                       :basePrice 85
                                       :View 30
                                       :Hot_Tub 20
                                       :Internet_Access 20
                                       :Kitchen 20}
                   :ordered_counts (3
                                    3
                                    2
                                    1)
                   :num_found 4
                   :ordered_values (155
                                    147
                                    151
                                    139)
                   :class_total 9}
   :count 9
   :filters {c constraints
               :smoking 0}
}
```

Note the value of the *:ordered_value* key is a list of numbers, representing the different prices of rooms that have been found. If we wish to talk about the range of prices in our generated string, we first need to sort the list. This task can be handled in the preprocessor, which is the component that performs any necessary processing to the frame before it reaches the grammar rules for generation.

price	*($if :num_found == "1" || :num_found == "2" >do_nothing >sort_prices)*
sort_prices	*$sort_items*

The above rules in the preprocessor file states that if there are more than 2 different prices in the rooms found thus far, then we will speak about the range of prices in the generated string. *$sort_items* is a command that sorts the list of numbers and appends the sorted list in the frame as the value of the *:sorted_values* key. The resulting frame is shown below. The final target string generated from this frame is *"I have found 9 nonsmoking rooms. The prices of the rooms range from 139 to 155 dollars. The base price of a room is 85 dollars, A view costs an extra 30 dollars, a hot tub 20 dollars, internet access 20 dollars, and a kitchen 20 dollars."*

```
{c singled_out_category
   :speak_about {c price
                   :pricing_policy {c pricing_policy
                                     :basePrice 85
                                     :View 30
                                     :Hot_Tub 20
                                     :Internet_Access 20
                                     :Kitchen 20}
                   :ordered_counts (3
                                    3
                                    2
                                    1)
                   :num_found 4
                   :ordered_values (155
                                    147
                                    151
                                    139)
                   :class_total 9
                   :sorted_values (139
                                   147
                                   151
                                   155)}
   :count 9
   :filters {c constraints
               :smoking 0}
}
```

5.2.2 Example 2: The *set* command

In this example, we demonstrate a useful way to use the *set* command other than for the purpose of adding key/value pairs in the frame for generation. Consider another *singled_out_category* frame below, whose generation string will prompt the user to choose between a smoking or non-smoking room when there is a choice.

```
{c singled_out_category
   :speak_about {c smoking
                   :ordered_counts (59
                                    43)
                   :num_found 2
                   :ordered_values (1
                                    0)
                   :class_total 102}
   :num_found 102
   :filters {c constraints
               :with "hot tub"
               :vacant "1"}}
```

The key *:ordered_counts* is a list of two numbers: the first number refers to the number of non-smoking rooms, and the second number refers to the number of smoking rooms that have been found. To generate an English string about smoking/nonsmoking options, we use the following grammar rules to check the values of *:ordered_counts* and branch to different generation rules depending on their values.

```
:ordered_counts    ($if :first >check_nonsmoking >do_nothing)
                   ($if :last >check_smoking >do_nothing)
check_nonsmoking   ($if :first == "0" >all_smoking >set_flag1)
check_smoking      ($if :last == "0" >all_nonsmoking >set_flag2)

set_flag1          ($set :flag1 "1")
set_flag2          ($set :flag2 "1")

check_both         ($if :flag1 == "1" && :flag2 == "1" >do_both >do_nothing)
do_both            !have_both_smoking_nonsmoking .
all_smoking        !all_smoking .
all_nonsmoking     !all_nonsmoking .
```

If the first number is *"0"*, then we only have smoking rooms, and the target string will be *"There are only smoking rooms"*. Otherwise, we set the value of the key *:flag1* to be *"1"* and proceed to check the second number. If the second number is *"0"*, we know all the rooms are nonsmoking rooms. Otherwise, we set the value of the key *:flag2* to be *"1"*. At this point, we either have already generated an English string, or both flags have been set to *"1"*. In the *check_both* rule, we generate a string stating there are both smoking and nonsmoking rooms if both flags have values equal to *"1"*.

From the above frame, Genesis generates the following sentence: *"I have found 102 rooms with a hot tub. There are both smoking and non-smoking rooms."*

5.3 Chinese Generation for Simulated Dialogues

The Chinese simulated dialogues are generated from the same semantic frames that we used to generate the English dialogues. Therefore, we are able to use the hotel-domain catalogs developed for English as a basis to modify and tailor to produce the Chinese utterances.

5.3.1 Chinese Inquiries

Due to the simplicity of the inquiry frames, the grammar rules for the Chinese inquiries did not need to be drastically altered aside from re-arranging the order of the sentence structure. The majority of changes occur in the lexicon files, where we have provided Chinese strings for the default generation strings.

A notable difference for the Chinese lexicon file is that the *Cycle* feature was not used to provide multiple paraphrases for the same semantic frame. We believe that, because the Chinese dialogues exist to provide a reference for the student in their native language, the translations should be consistent from one simulation run to the next.

5.3.2 Chinese Responses

The responses of the dialogues are generated from meaning representations that are more complex structurally and contain a good amount of linguistic information. With the sentences being longer and richer in meaning, the grammatical differences between Chinese and English becomes apparent. Consequently, the catalog for the Chinese responses needed significantly more work to port from the English response catalogs.

For example, predicates are used differently in English and Chinese. The following sections of a semantic frame represent the strings *"a room with a view"*, and *"a room* *with a kitchen"*.

```
:filter {c constraints
          :with "view"
          :vacant "1"}

:filter {c constraints
          :with "kitchen"
          :vacant "1"}
```

These phrases both have *"with"* as the predicate. However, as shown in the following table, the predicates for the corresponding Chinese sentences are *"see"* and *"have"*, respectively. These issues needed to be resolved in the Chinese catalogs, as we are generating Chinese strings from a semantic frame that is intended for English language generation.

ke3 yi3	*kan4 dao4*	*feng1 jing3*	*de5*	*fang2 jian1*
able	*see*	*view*	*'s*	*room*

you3	*chu2 fang2*	*de5*	*fang2 jian1*
have	*kitchen*	*'s*	*room*

5.4 Translating English to Chinese

Up until this point, we have generated Chinese strings for the simulated dialogues from either User Simulator or Turn Manager frames. Next, we will motivate the need for creating a Genesis catalog to provide Chinese translations directly from English sentences.

We envision that, in the language learning system, if the user has trouble speaking a sentence in English to the system, he or she can say it in Chinese, and the system will speak the corresponding English sentence back to the user. For the system to understand and parse the student's Chinese utterance, a working Chinese recognizer is needed. We can create a Chinese grammar by writing the grammar rules by hand, but this is a tedious process. Instead, we collaborate with another ongoing project in the SLS group to produce a Chinese grammar.

A current project in the SLS group is to induce a grammar for a target language from an English grammar using a set of English sentences and its translation in the target language. To use this method for acquiring a Chinese grammar, we created a Genesis catalog that is able to translate a set of sentences from English to Chinese within the hotel domain.

We use the Chinese catalog of Jupiter - a weather information system, as a starting point for our Chinese catalog for the hotel domain. The translation process is defined as the procedure in which an English sentence is parsed into a meaning representation by TINA with an English grammar, and then passed to Genesis to generate a Chinese string using the appropriate linguistic catalog. Since the incoming frames to the language generator are not domain specific - as has been the case for other catalogs described thus far, the grammar rules are required to accurately translate sentences within the hotel domain, while preserving generality so that they may be expanded to handle any other input frame produced by TINA.

We divide the grammar file into 3 separate sections for clause, predicates and topic rules. The grammar rules are designed to manage input frames as a group by using group rule- templates rather than having individual rules to target each kind of frame. When new frames are introduced, we look for a rule-template that is appropriate for generating the new frame, and add the frame name to the corresponding group list. If there does not exist a rule that will correctly handle the new frame, then a new group rule-template is added to the grammar file.

For example, predicates are grouped by types according to the position in which they can appear in a sentence. *Prepreds* are the group of predicates that appear before a topic (such as *adjective*, *locative*), and the group *preds* contains the predicates that can appear elsewhere. Within these groups, the predicates are further divided into subgroups, each of which has a group rule-template. The following is a rule that handles all *wh_complement* frames. Notice that *>prepreds* and *>preds* are in the appropriate locations of the resulting target string. If the *wh_complement* frame contains any sub-frames that belong in one of the predicate groups, it will be processed using the appropriate template rule in the specified order.

wh_complement	*>prepreds :topic :auxil >preds*
prepreds	*>main_prepreds for >prepreds_with >prepreds_de >prepreds_core*
preds	*>predless_preds >other_preds*
main_prepreds	*quality month_date temporal time_interval* *locative crisis_type open on at about besides* *possess adjective_phrase current specific special* *extended general other prep_phrase*
main_prepreds_template	*:adv (:topic $core)*

.
.

5.5 English Strings from Chinese Parses

To evaluate the accuracy of the induced Chinese grammar, we use it to parse a set of hotel related utterances in Chinese and compare it with the parse of the corresponding English utterance parsed with a correct English grammar. We find that the induced Chinese grammar produces parse trees that are the same as the English parse tree with the exception of two points. The first is that quantifiers such as *"a"* and *"the"* are missing from Chinese parses, as these do not exist in the Chinese language. The second issue is the confusion between *verify_questions* and *wh_questions* in the Chinese grammar.

Overall, the meaning representation produced by the induced Chinese grammar achieves high accuracy. Using an existing generic English catalog - with minor modifications to the preprocessor file to add appropriate quantifiers, we were able to generate well-formed English sentences with the Chinese semantic frames as the input.

5.6 Envoice Shortcuts

Lastly, we turn to Genesis to provide a solution for improving the speech synthesis quality. In many cases, a word or a phrase needed by the synthesizer to form new sentences exists in more than one place in the recorded speech corpus. Occasionally, Envoice selects word units for concatenation from a place that results in less than ideal synthesis. Rather than exploring concatenation constraints for word selection, we use Genesis to specify a "shortcut" to the best available waveform so that Envoice will always pick the optimal selection for synthesis.

Suppose we'd like to synthesize a waveform for the sentence *"Oh and I'd like a wake up call tomorrow at nine am"*. If we allow Envoice to use its scoring algorithm to select the waveform segments, we observe that the waveform achieves high quality except for the word *"at"*, which is not complete and sounds distorted. If there is another place in the recorded speech corpus where *"at"* appears and is a better choice, we can use the *$:envoice* tag to specify that this *"at"* be chosen every time this sentence is synthesized. In the lexicon file, we can specify the file where the targeted word is located, along with the begin/end time that will extract the word.

wakeup_call	*O*	*"oh and i'd like a wake up call tomorrow" !at :time*
at	*O*	*"at" $:envoice "[hotelroom/wavs/hotelroom_user173 31880 34120 1 1 1 at]"*

We employ this method where necessary to complete the refining process of the synthesized speech.

Chapter 6

Evaluation

6.1 Evaluation of Language Generation

The task of effectively evaluating natural language generation (NLG) is a difficult one. Numerous literature have discussed the various issues with evaluating a NLG system. The main concern is the lack of well-defined input and output [3]. It is often difficult to judge whether the input to the system is actually "cheating" by including some form of guidance to the system on how to handle specific problems. Using input that is generated by a separate system unrelated to NLG for evaluation purposes would be ideal [5]. However this is not a feasible approach for many systems, including ours, as the meaning representation input to our system is constructed specifically for the generation task at hand.

It is also difficult to obtain a quantitative, objective way to measure the quality of the output text. To assess the quality of a generated text based on criteria in such as *Accuracy*, *Fluency*, and *Lexico-grammar coverage*, methods of evaluation can be categorized in the following three classes [1, 7].

Intrinsic evaluation: This typically consists of using human judges to rate the generated text. It is the most straightforward method and the easiest to carry out. The problem with this method is that evaluation is very subjective to the human evaluator's personal style and preference.

Extrinsic or task evaluation: This method measures the quality of the text by assessing the user's ability to perform some task according to the generated text. This method of evaluation is more difficult to execute, as it requires more experiments to complete.

Comparative evaluation: The goal here is to directly compare the performance to a different generation system. Comparative evaluations is not often practiced due to its complexities.

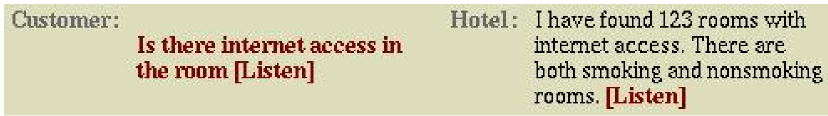
6.1.1 Intrinsic Evaluation

To evaluate the language generation in this thesis, we use human judges to critique the quality of the generated text. In addition, we utilize Microsoft Word's spell and grammar check to give a rough overall assessment. We rate the generated text on the following criteria: *correctness*, *coherence*, *content* and *organization*.

Correctness refers to the accuracy of the text based on the content of the meaning representations. That is - is everything in the meaning representation that is intended to be spoken generated and generated correctly? This is something that can be checked objectively without relying on outside evaluators. All of the generated sentences accurately and completely reflect the relevant information in the meaning representations. Simulation logs containing several simulated dialogues in both English and Chinese were given to five people. Two of them are native speakers of Chinese and are fluent in English; one is native in English and fluent in Chinese; the other two are native in English and did not evaluate the Chinese generation. Evaluators were asked to read the generated text and remark on coherence, content, organization, and provide any other comments they may have.

As expected, there were discrepancies among the evaluations. Overall, the responses were positive and encouraging. All stated that the generated text - both English and Chinese - were natural, coherent, well formed, and contained only trivial faults - if any. We will now present some of the evaluators' constructive feedbacks.

Evaluator A remarked that she was impressed with the amount of information the reply sentences were able to convey, and how well put the sentences were. However, Evaluator B thought those same reply sentences contained slightly too much information. He commented that some of the sentences would have been better phrased as questions. Consider the following snippet from a sample conversation:



Evaluator B said that the sentence *"There are both smoking and nonsmoking rooms"* was a somewhat unanticipated and irrelevant given the customer's question, and would have been better phrased as a question *"Would you prefer a smoking or nonsmoking room?"*

The reason this option was posed as a sentence rather than a direct question is that the system wants to provide some guidance, but does not want to in any way restrict the user's next utterance. The user can choose between a smoking or nonsmoking room, or he can request something entirely different - even contradictory to what had been said before - and the system is capable of handling that and continuing with the conversation.

Another noteworthy observation is with regards to the Chinese reference for the simulated dialogues. Both English and Chinese dialogues were constructed from the same semantic frames. We used the *Cycle* feature in the lexicon file to provide multiple English generations from the same meaning representation. This causes some disparity between the English and the Chinese translations in certain cases since one English paraphrase can be literally closer to the Chinese text than another. For example, the English sentence *"Can I reserve a room next Thursday for two days"* may have, as its Chinese translation, *"Can I reserve a room for next Thursday and Friday"*.

Evaluator C felt that overall, the generated Chinese text were of high quality - coherent, and comparable to human crafted text. However, he noted that a few Chinese inquiries were slightly awkwardly phrased, and would sound more natural if phrased in a different way that is less of a literal translation from the English.

From the human evaluations, we were able to confirm that the quality of our language generation system achieve a respectable level. At the same time, we also received constructive criticisms that will help us a great deal in refining the system.

6.1.2 Microsoft Word: Spell and Grammar

Microsoft Word (MS Word) is a prevalent word processor choice for most people. One useful and advanced feature of MS Word is the spell and grammar check. While it is true that MS Word does not catch all errors in a text document and sometimes falsely identifies correct phrases as errors, it does a good job overall in checking a document for syntactic and grammatical errors. We used MS Word's spell and grammar check on our simulate dialogues and the results were virtually error free. The only complaints MS Word had were some spacing and capitalization issues.

6.2 Evaluation of Speech Synthesis for Simulated Dialogues

6.2.1 Method

We aim to take an objective approach to evaluate the speech synthesis for the simulated dialogues. For a random set of 100 synthesized waveforms, we listen and categorize them into one of three clearly defined classes. **Near perfect** refers to the waveforms that sound as if the utterance was recorded in whole rather than synthesized from separate segments. **Adequate/Satisfactory** are waveforms that are noticeably synthesized from different waveform fragments. All words in the sentence are clear and complete, with no significant silent gaps in between words, and where tone and pitch of words deviate only slightly. **Improvement Needed** includes waveforms that contain segments that are inaudible, incomplete, incorrect in tone or pitch, and overall unsatisfactory.

6.2.2 Results

We listened to one hundred synthesized waveforms and judged them according to the above criteria. We found **81** of them to be **Near Perfect**; **18** fell in the class of **Adequate/Satisfactory**; and one waveform - an utterance consisting of two words with the wrong pitch, belonged to the **Improvement Needed** category.

We are very pleased with these results. Refining the recorded corpus waveforms with manual alignment appears to have contributed notably to the end results. Nearly all words in the synthesized waveforms are crisp and complete. Currently, the majority of the flaws are in the tone and pitch of some words. We believe that recording additional sentences to add to the speech corpus, so that we can extensively cover all possible prosodic contexts, would further enhance the quality of the audio output.

Chapter 7

Conclusion

7.1 Summary

The goal of this thesis is to further expand and improve conversational capabilities of the initial version of the SLLS. Mainly, we focused on generating natural, well formed, grammatically correct English sentences and producing high quality synthesized waveforms for language students to emulate. Our work is centered around a language learning system for native Chinese speakers learning English as a foreign language. Presently, we concentrate on conversations within the hotel domain.

The goals listed in Chapter 1.2 were successfully accomplished. This thesis makes extensive use of Genesis to complete a variety of natural language generation and translation tasks. To highlight, we are able to generate sample dialogues in both English and Chinese, with each dialogue specific to a distinct hotel simulated by a Random Hotel Generator. We developed a Genesis catalog to accurately translate phrases in the hotel domain from English to Chinese. Using this set of translations and an existing English grammar, we utilize the technology of another project in the SLS group to obtain a Chinese grammar to correctly parse hotel-related Chinese utterances.

Using Envoice, we produced high quality synthesis for two voices, to be used as the voice of the customer in simulated dialogues. Genesis also played a role in refining the quality of the speech synthesis.

7.2 Future Work

We believe that an interactive learning system is a good model for language learning. Integrating the work that was completed in this thesis, with the existing SLLS web-based infrastructure will result in a system that is ready for trial by language students. At that point, it will become feasible to run user studies to determine whether the system truly is helpful for language learning.

We expect to expand the conversational topics of SLLS to cover other domains. In order to do this effectively, it is worthwhile to explore methods for developing a domain-independent dialogue manager, such that the same code can be reused for other domains at later times. Preserving generality in other components, such as the user simulator and language generation module, also allows flexibility and ease for incorporating new domains in the future. Once a system is complete and stable, a next step may be to port the system to support the learning of another language.

Thus far, we have used the interactive environment to target improving one's speaking skills. The technologies at SLS can be modified to focus on writing skills as well. For example, a current research project in SLS supports typed-input drill exercises for students learning Mandarin.

Lastly, language learning is only one application of the conversational capabilities in SLS. A wide variety of spoken language applications, such as weather information, airline flight planning/status, and restaurant search have already been developed. Similarly, our hotel-domain language learning system can be adapted into an automated hotel search service, a resource that would be beneficial to many people.

Appendix A

Reference Guide to Genesis Commands

- **Clone**
  - ($clone *:source[:keyword]*)

    equivalent to ($set *:source :source[:keyword]*).

    Syntactic sugar for the *set* command.
- **Core**
  - *$core*

    Generates vocabulary for the current frame's name, and adds the result to the target string.
- **Gotos**
  - >*grammar_rule*

    Descends into *grammar_rule*, executes it, adds the result to the target string, and continues with generation in the original rule.
- **If/Else**
  - *($if :keyword then_command else_command)*

    If the *if* command evalutes to true, then the *then_command* is executed, and the result is added to the target string. Otherwise, the *else_command* is executed. The *else_command* is optional.

- **keywords**
  - *:keyword*

    Searches the current frame for the *:keyword*, processes its key value, and add the result to the target string.
- **List**
  - *:nth*

    Identifies each list item.
  - *:first*

    Identifies the first item in the list.
  - *:butlast*

    Identifies each item but the last in the list.
  - *:last*

    Identifies the last item in the list.
  - *:singleton*

    Identifies the item in a singleton list.
- **Lookups**
  - *!string*

    Generates vocabulary for the *string*, and adds the result to the target string.
- **Pull**
  - *−−deferred_string*

    Searches the *info frame* for the *−−deferred_string*. If found, adds the key value of the *−−deferred_string* to the target string.

- **Push**
  - $>$ $--$*grammar_rule*

    Descends into the *grammar_rule*, executes it, defers the resulting string by adding it to the *info frame* as the key value for $--$ *grammar_rule*, and continues generation in the original rule.
- **Or**
  - *(command1…commandN)*

    Executes each of the commands sequentially until one produces a string, then adds the result to the target string.
- **Predicates**
  - *predicate*

    Searches current frame for the *predicate*, processes it, and adds the result to the target string.
- **rest**
  - *$rest*

    Generates a string for all predicates in the current frame that have not yet been processed, and adds the result to the target string.
- **Selectors**
  - *grammar_rule command1…$:selector…commandN*

    sets the *$:selector* in the *info frame*.
- **Set**
  - *($set :target source)*

    Generates the target string for *source*, and adds the result to the current frame as the key value for *:target*. *source* can be a string, a lexicon lookup, a keyword in the current frame or in a child frame of the current frame.

- **String**
  - *"string"*

    Adds the *string* to the target string.
- **Time**
  - *$time*

    Preprocesses the current frame as a time frame and attempts to add key values for *:hours*, *:minutes*, *o+clock*, and *:xm* .
- **Tug**
  - < −−*:keyword* or < −−*predicate*

    If children frames contain *:keyword/predicate*, moves the *:keyword/predicate* and its key value into the current frame, generates a string, and adds it to the target string.
  - < −−*:keyword[key1 pred1 pred2...]* or < −−*predicate[key1 pred1 pred2...]*

    If the current frame contains a childframe matching one of the predicates or keyword in the bracket and if the child frame contains the *:keyword/predicate*, moves the *:keyword/predicate* and its key value into the current frame, generates a string for them, and adds it to the target string.
  - < −−*grammar_rule[key1 pred1 pred2...]*

    If the current frame contains a child frame matching one of the predicates or keywords in the brackets, generates a string for the child by using the *grammar_rule*, and adds the result to the target string.
- **Yank**
  - *<==command*

    Idential to the *tug* command except that the *yank* is not restricted to searching only the children of the current frame. The *yank* command performs a breadth-first search of all descendants.

Bibliography

[1] S. Bangalore, A. Sarkar, C. Doran, and B. A. Hockey. Grammar and parser evaluation in the XTAG project. In *The Workshop on Evaluation of Parsing Systems*, Granada, Spain, 1998.

[2] L. Baptist. Genesis-II: A language generation module for conversational systems. Master's thesis, Massachusetts Institute of Technology, September 2000.

[3] K. Bontcheva. Reuse and challenges in evaluating language generation systems: Position paper, April 2003. Accessed online at http://www.dcs.shef.ac.uk/ katerina/EACL03-eval/eacl-doc/Bontcheva.pdf.

[4] S. Crockett. Rapid configuration of discourse dialog management in conversation systems. Master's thesis, Massachusetts Institute of Technology, 2002.

[5] R. Dale and C. Mellish. Towards the evaluation of natural language generation. In *The First International Conference on Evaluation of Natural Language Processing Systems*, Granada, Spain, May 1998.

[6] J. Lau. SLLS: An online conversational spoken language learning system. Master's thesis, Massachusetts Institute of Technology, May 2003.

[7] N. Miliaev, A. Cawsey, and G. Michaelson. Applied NLG system evaluation: FlexyCAT. In *The Tenth Conference of the European Chapter of the Association of the Computational Linguistics*, April 2003.

[8] Massachusetts Institute of Technology. Spoken language systems group. http://www.sls.csail.mit.edu.

[9] A. Ratnaparkhi. Trainable methods for surface natural language generation. In *The First Language Technology Joint Conference of Applied Natural Language Processing and the North American Chapter of the Association for Computational Linguistics (ANLP - NAACL2000)*, Seattle, Washington, April-May 2000.

[10] E. Reiter. NLG vs. templates. In *The Fifth European Workshop on Natural-Language Generation*, Leiden, The Netherlands, 1995.

[11] S. Seneff. TINA: A natural language system for spoken language applications. *Computational Linguistics*, 18(1):61–68, March 1992.

[12] J. Yi. Natural-sounding speech synthesis using variable-length units. Master's thesis, Massachusetts Institute of Technology, May 1997.