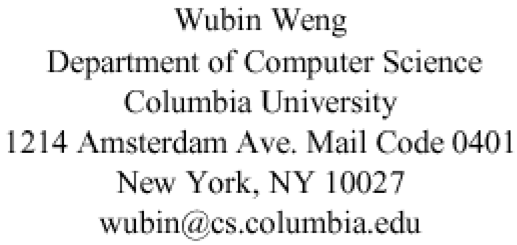
**RC22420 (W0204-139) April 30, 2002**
**Computer Science**

# IBM Research Report

## Designing a speech corpus for instance-based spoken language generation

**Shimei Pan**
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598



Wubin Weng
Department of Computer Science
Columbia University
1214 Amsterdam Ave. Mail Code 0401
New York, NY 10027
wubin@cs.columbia.edu

# Designing a Speech Corpus for Instance-based Spoken Language Generation

Shimei Pan
IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
shimei@us.ibm.com

Wubin Weng
Department of Computer Science
Columbia University
1214 Amsterdam Ave. Mail Code 0401
New York, NY 10027
wubin@cs.columbia.edu

## Abstract

In spoken language applications such as conversation systems where not only the speech waveforms but also the content of the speech (the text) need to be generated automatically, a Concept-to-Speech (CTS) system is needed. In this paper, we address several issues on designing a speech corpus to facilitate an instance-based integrated CTS framework. Both the instance-based CTS generation approach and the corpus design process have not been addressed systematically in previous researches.

## 1 Introduction

Instance-based (concatenation-based) Text-to-Speech (TTS) synthesis in which pre-recorded speech segments are reused and concatenated to generate new utterances becomes increasingly popular. So far, the best TTS systems available on the market are instance-based. In addition, instance-based approaches are particularly effective for domain-specific applications in which large phrases, sometimes even entire spoken utterances can be reused for both waveform and prosody generation (Donovan, 1999, Pan, 2002).

In many spoken language applications such as conversation systems where not only the speech waveforms but also the content of a speech (the text) need to be automatically generated, a TTS system is not sufficient because it requires online text as input. Instead, we need a Concept-to-Speech (CTS) system in which speech is generated directly from semantic representations. There are two separated stages in a traditional CTS framework, natural language generation (NLG), which constructs the content and produces grammatical text, and TTS, which synthesizes speech from the text. In this uncoupled framework, a TTS has to infer everything such as syntatic and discourse context from text, even though this information is available during NLG. Since not only pre-recorded speech segments but also sentence structures and wording can be learned and reused based on pre-stored corpus instances (Varges 2001), we extend the same framework to cover the entire CTS process. Moreover, since instance-based approaches work well in domain-specific applications, and almost all the existing CTS applications are domain-specific, we expect this approach to be effective for most CTS applications. Overall, instance-based learning provides a general platform for integrated text and speech generation. In such a CTS system, the decisions in text generation directly affect speech synthesis, which is difficult to achieve in traditional CTS systems.

To facilitate an instance-based CTS framework, we create a speech corpus from which our system learns both text generation and speech synthesis. Since all the linguistic and speech knowledge used by the system is encoded in the corpus, what is available and how information is represented in the corpus have direct impact on the capability of a CTS generator. Until now, research issues on corpus design for empirical-based CTS generation have not been systematically addressed.

Our work is part of a larger effort in developing multimodal conversation systems. To aid users in their information-seeking process, we are building an intelligent infrastructure, called Responsive Information Architect (RIA), which can engage users in a full-fledged multimodal conversation. A user interacts with RIA using multiple input channels, such as speech and gesture. Similarly, RIA acts/reacts to a user's request/response with automatically generated speech and graphics presentations. Currently, RIA is embodied in a testbed, called Real Hunter, a real-estate application for helping users find residential properties. As part of the effort, we are building SEGUE (Spoken English Generation Using Examples), the CTS generator in RIA. SEGUE employs an instance-based framework to systematically generate both text and speech.

The rest of the paper is organized into three sections. We first describe the principles and an algorithm used in collecting and generating corpus instances. Then we describe the annotation schemas represented in XML format, which capture typical language and speech features. Finally, we briefly discuss how this corpus is used in CTS generation.

## 2 Designing Corpus Scripts

When we prepare scripts to be read during recording, we pay attention to not only the content of the scripts but also the syntactic and surface properties of the text because they will affect the assignments of some critical spoken language features, such as prosody, when the text is read. Our design principles cover several aspects of the script design process.

### 2.1 Design Principles

The general principle is to create a corpus to cover words and sentences that are most likely to be reused. In addition, a corpus should also cover sufficient variations to support flexible and natural spoken language generation. Thus, each instance in a corpus should fulfill at least one of the following purposes: improving the semantic coverage, improving the syntactic coverage, improving the prosodic coverage, and improving the word coverage.

***Semantic coverage:*** In the corpus, there should be at least one instance covering each domain concept and relation independently. For example, in RIA, we cover all the concepts and relations represented in RIA's domain ontology.

***Syntactic coverage:*** The corpus should cover rich syntactic variations similar to those observed in natural language. Syntactic paraphrases not only create less repetitive but livelier sentences, but also provide rich substructures (such as noun or adjective phrases) to be reused in constructing new sentences.

***Prosodic coverage:*** A corpus should cover as many prosodic variations of the same words or phrases as possible because during speech synthesis, a CTS system looks for prosodically appropriate speech segments to reuse. Unlike in (Theune, 2001) where six different prosodic realizations are used for each variable based on combinations of accents and prosodic phrase boundaries, we use a different approach because only after the final scripts are read and recorded will the exact prosody of the corpus instances be determined, Thus, during our script preparation stage, prosodic variations are indirectly controlled by carefully varying the syntactic and surface properties of the corpus instances.

***Word coverage:*** A corpus should cover as many domain words as possible even though full word coverage is hard to achieve.

Of all the four design principles, only the semantic coverage is required. For the other three principles, the better the coverage is, the better the generation quality is. Since word coverage is not mandatory, during generation, the system may not be able to find the appropriate words/phrase to use. In this case, the system will fall back to a general TTS to generate missing words/phrases.

In the following, we focus on how to collect corpus instances to satisfy these requirements.

### 2.2 Resources

When we prepare scripts to be read, the scripts should be close to the utterances to be generated, both in term of content and style. Since RIA responses to a user with automatically generated speech and graphics presentations, ideally all the corpus scripts should come from multimodal conversations. In addition, the corpus should also cover the real estate domain. So far, there is only another multimodal conversation corpus known to us that covers the real estate domain (Yan, 2000). But its content is quite different from what we need. For example, it focuses on the floor plan and spatial description of a house, while we also need to cover town, school, and transportation information. Our final corpus material comes form several different resources.

First, there is a large amount of real estate data available online. Online web sites are RIA's main source of information. However, they contain primarily written texts and sometimes, the content may not be appropriate for speaking. In designing a corpus that is appropriate for speech, we gathered speech transcripts from real estate TV programs. We also collected both unimodal and multimodal conversation transcripts for RIA. So far, we have transcripts from our initial user study. We also added multimodal conversation scripts from both RIA's test runs and mock-up demos. Thus, our *initial corpus* consists of scripts from several different resources. In the following, we describe how to transform such an *initial corpus* into one that is compatible with our previsouly mentioned design principles.

## 2.3 Creating carrier sentences

Carrier sentences were created to encode different sentence patterns. For example, the carrier sentence for "*This colonial home is at Pleasantville*" is "*This $STYLE home is at $TOWN*", Where *$STYLE* and *$TOWN* are variables. Each variable may take one or more values. For example, the values for *$STYLE* includes *colonial* and *contemporary*. The main reason for using carrier sentences instead of the instances themselves is that each variable in a carrier sentence can later be instantiated with different values to create new corpus instances. In the following, we first focus on constructing carrier sentences. Then, in the next section, we describe an algorithm for duplicating and instantiating carrier sentences.

For each sentence in the original corpus, we systematically use variables that represent domain concepts to replace those words realizing the concepts. In addition, we remove repetitive carrier sentences because later we systematically duplicate carrier sentences based on the design principles.

Since carrier sentences are collected from a variety of resources prepared by humans, they cover the general semantic concepts of the domain. In addition, they also encode natural syntactic variations in human language. But in some cases, some domain concepts are still missing in the corpus. To convey those uncovered domain concepts, we specifically construct new carrier sentences. In addition, whenever possible, we add new paraphrases for carrier sentences to increase syntactic variations.

## 2.4 Instantiating carrier sentences

During carrier sentence creation, we focus on semantic and syntactic coverage. During instantiation, however, our primary concerns are prosodic and word coverage.

The instantiation process can be seperated into two steps: enumerating possible values for each variable and duplicating/instantiating carrier sentences. we illustrate them one by one.

**Enumerating values :** Before a carrier sentence can be instantiated, we need to know the possible values of each variable. Ideally, we should cover all the possible values for each variable to ensure proper coverage. In practice, the values of some variables are either impossible to enumerate or too large to enumerate effectively. Among them, proper names and numerical variables pose the biggest challenge.

In terms of proper names, such as person names, the possible values are too large to enumerate. Thus, one typical strategy is to cover the most common proper names and hopefully they will cover most names to be generated.

For numeric variables, such as zip codes, and house prices, it is impossible to list all the values. However, word coverage is not difficult to achieve. For example, for zip codes, ten digits will be enough. For house prices, numbers from one to nineteen plus twenty, thirty, to ninety and plus million, thousand, hundred will be sufficient. In addition to word coverage, because the same digit may sound different in speech, prosodic coverage is also a concern. For example, the 1s in the zip code 10511 may all sound differently due to prosodic variations. Thus, it is a good idea to cover all the 1s in different positions in zip codes. A typical solution for generating critical values for numeric variables is to analyze the prosodic patterns of each variable and cover not only the digits but also each prosodic realization of the digits. In our zip code example, the goal is to have all the digits appear in each position at least once because they may have different stress patterns. Only ten zip codes are needed for both word and prosodic converage : 12345, 23456, 34567, 45678, 56789, 67890, 78901, 89012, 90123, 01234. Simiar approaches were also used in (Yi, 1998 ). Ideally, we should also consider co-articulation. However, it will produce too many combinations. One way to alleviate the influence of co-articulation is by instructing the speaker who reads the scripts to put a pause between numbers.

**Duplicating and instantiating carrier sentences:** The main goal in duplication is to generate enough carrier sentences so that we have sufficient number of instances to cover all the essential values of each variable at least once. Moreover, because words in different positions may associate with different prosodic patterns, we also want to make sure that each value also appears in every position at least once in order to increase prosodic variations. For example, the word *colonial* in *This colonial house is in Pleasantville* may sound differently from the *colonial* in *The style of the house is colonial*. Thus, during instantiation, we want all possible house styles appear in each of the two places at least once. Currently, we categorize all the possible sentence positions into three classes : sentence initial, sentence middle and sentence final. The main reason for this generalization is to reduce the number of instances needed to cover position variations. However, fine-grained classifications may produce better results if the total number of instances is not a concern. In addition, we also want to duplicate as few carrier sentences as possible to control the overall corpus size because manual annotation is often needed for corpus-based CTS generation. Thus, we want each carrier sentence simultaneously serves as many purposes as possible.

One way to instantiate carrier sentence is to use a Context Free Grammar (CFG)-based generation approach. But this may generate too many instances. For example, given two carrier sentences : *This $STYLE house is in $TOWN* and *This $TOWN is the home of this $SYTLE house*, and two values for each variable : *colonial and contempory* for $STYLE, *Pleasantville and New Castle* for $TOWN, the CFG-based approach generates eight instances while only four instances are enough to have both the position and word coverage

*Colonial+Pleasantville*
*Colonial+New Castle*
*Contemporary+Pleasantville*
*Contemporary+New Castle*
*Pleasantville+Colonial*
*Pleasantville+Contemporary*
*New Castle+Contemporary*
*New Castle+Colonial*

**CFG output**

*Colonial+Pleasantville*
*Contemporary+New Castle*
*Pleasantville+Colonial*
*New Castle+Contemporay*

**Ideal output**

The algorithm we proposed here accomplishes the desired word and position coverage with less carrier sentences than the CFG-based approach. Figure 1 show the pseudo code for duplicating and instantiating the carrier sentences :

```
Duplication:
(1) For each variable $V_i$ in the domain :
    $N_{vi}$ = the number of possible values of $V_i$
    1.1 for all the carrier sentences $C_j$ in the corpus
        First($V_i$) = the number of $C_j$ where $V_i$ is at the begining.
    1.2 for all the carrier sentences $C_k$ in the corpus
        If First($V_i$) >= $N_{vi}$ Then $N_d$=0
                                  Else $N_d$=Round($N_{vi}$/First($V_i$))-1
        If $V_i$ appears first in $C_k$ Then Duplicate $C_k$ for $N_d$ times
    Repeat Step 1.1 and 1.2 for $V_i$ at sentence middle and final positions on the new corpus.
 Repeat (1) for all the variables in the domain on the new corpus.
Instantiation:
(2) For each variable $V_i$ in the domain
    2.1 $LIST_{vi}$=the list of all the possible values of $V_i$
    2.2 For all the carrier sentences $C_j$ in the corpus
       If $V_i$ is at the beginning of $C_j$ Then $Va$=rotate( $LIST_{vi}$)
                                              Replace $V_i$ in $C_j$ with $Va$
       Repeat step 2.2 for $V_i$ at sentence middle and final position.
Repeat (2) for all the concepts in the corpus
```

**Figure 1: An Algorithm for Instantiating Carrier Sentence**

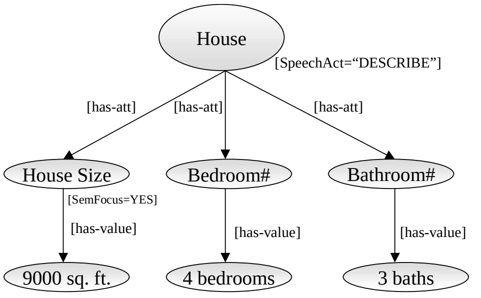
In Figure 1, $V_i$ is the current variable, $C_j$ and $C_k$ are the current carrier sentence, $N_{vi}$ and $N_d$ are the number of possible values for variable $V_i$ and the number of duplications needed for a carrier sentence. For each variable at each sentence position, we check whether there is sufficient number of carrier sentences to cover all the values of that variable. If the answer is yes, no duplication is needed. Otherwise, the system computes how many more carrier sentences are needed and duplicate carrier sentences evenly across all the related carrier sentences. In addition, each time new carrier sentences are created, they are put back to the corpus so that the computation for a different variable at a different position will take these new carrier sentences into consideration. This is one way to avoid generating too many carrier sentences. In step (2), the system replaces each variable with its values systematically.For SEGUE, so far we have collected and created over 300 carrier sentences. After duplication and instantiation, there are about 1000 instances in the final corpus. In the following, we describe how each corpus instance is annotated to facilitate instance-based CTS generation.

## 3 Corpus Annotation

For simple applications, carrier sentences themselves have been used directly in corpus-based NLG (Ratnaparkhi, 2000). However, they may not be comprehensive enough for more sophisticated applications. For example, unlike domain concepts, relations are not explicitly annotated in a carrier sentence. Thus, given two concepts like $HOUSE and $TOWN as input, in principle, without indicating their relations, it is hard if not impossible to decide which sentence to choose: *$HOUSE is located in $TOWN* or *$HOUSE is close to $TOWN.* In addition, carrier sentences do not encode discourse influence, which may affect a CTS system's ability in generating coherent discourses. Moreover, carrier sentences do not encode the intentions of an utterance. Intentions are critical for conversation systems.

In SEGUE, we employ a comprehensive representation of corpus instances. Each training instance is associated with two annotations: a semantic graph (SemGraph) that represents the meaning of a sentence and a Realization Tree (ReaTree) that represents the syntactic, lexical, prosodic and acoustic realizations of the meaning. Both annotations are represented in XML format. In the following, we describe the features represented in these annotations.

### 3.1 Semantic Graph (SemGraph)



"This home has 4 bedrooms, 3 baths and 9000 square feet."

**Figure 2: A SemGraph**

Figure 2 shows a SemGraph for the sentence *This home has 4 bedrooms, 3 baths and 9000 square feet*. It encodes domain concepts, such as BEDROOM# and HOUSESIZE. In addition, it also encodes relations between concepts, such as HAS-ATT and HAS-VALUE. Overall, a SemGraph is an aggregation of domain relations and concepts. In addition, it also represents speech acts and semantic focus, which form the intention of a sentence. Currently, the speech acts covered in our annotation include *request, describe, confirm, help, greet, goodbye,* and *acknowledge.* Among them, *describe* and *request* are the most common speech acts in information-seeking applications. *Semantic focus* marks the attentional focus that a speaker wants to emphasize so that special syntactic constructions (e.g. preposing) or prosodic constructions (e.g. stress) can be used to realize the intention effectively. For example, if a speaker wants to emphasize that a house is huge, she may mark $HOUSESIZE the semantic focus.

### 3.2 Realization tree (ReaTree)

ReaTree encodes features related to how meanings are realized in speech. Since the same input can be realized in many different ways due to discourse, syntactic, lexical, prosodic, and acoustic variations, a ReaTree should cover all the relevant features.

The biggest challenge in encoding all these information in a ReaTree is that overall there are three different structures to be represented in a ReaTree : *a syntactic tree* encoding the syntactic constituent structure, *a semantic representation* encoding a SemGraph equivalence, and *a prosodic tree* encoding a prosodic constituent structure. Moreover, there is no simple one-to-one mapping between two different structures. For example, there is phonological evidence indicating that there is no direct mapping between a syntactic tree and a prosodic tree (Bechenko, 1990).

To solve this problem, the ReaTree representation is primarily based on a sentence's syntactic structure. On top of the syntactic tree, we use a set of features to mark the underlying semantic and prosodic structures. In addition, we also include features that are essential for discourse generation and speech synthesis. Here are the main features annoated in a ReaTree :

***Discourse feature:*** It encodes whether a syntactic constituent is the *topic* of a sentence. It is useful in generating context-appropriate sentences. For example, one strategy to maintain discourse coherence is to keep the current sentence *topic* the same as the discourse focus.

***Syntactic features:*** Main syntactic features annotated in a ReaTree include *syntactic constituent structures, syntactic categories (cat), grammar roles (role), syntactic functions (syn_fun)* and *part-of-speech (pos*). Syntactic features are used mainly for reconstructing new sentences. For example, *syntactic structures* are encoded as hierarchical syntactic trees. Each subtree or branch in a syntactic tree is a potential building block for new syntactic trees. In addition, *syntactic categories (cat),* such as whether a phrase is an NP, VP or ADJP, also help us decide whether two or more phrases can be combined to form a new phrase/sentence. *Grammar role*s, such as whether a constituent is a subject, object, or subject complement, provide more constraints on whether a syntactic constituent can be reused in a new sentence. *Syntactic functions* indicate whether a word or phrase is the *head* or *modifier* of a constituent. Finally, *part-of-speech (pos)* is the syntactic category of a word.

***Semantic features:*** For each syntactic constituent in a ReaTree, we also use the features like *base_concept* and *rel_concept* to encode the corresponding concept/relation realized by this constitute. Since the same concept/relation is also defined in the SemGraph, *base_concept* and *rel_concept* establish links between a concept/relation in a SemGraph and its realization in a ReaTree. Thus, it essentially defines a mapping between a SemGraph and the associated ReaTree.

***Lexcial feature :*** Right now, SEGUE only uses one feature called *text*, which is the exact wording used to convey a concept or relation.

***Prosodic features:*** The main prosodic features encoded in a ReaTree are the four main ToBI[1] features: *break index, pitch accent, phrase accent, and boundary tone* (Silverman 1996). *Break index (index)* describes the relative levels of disjuncture between two adjacent orthographic words. Five levels of disjuncture, form 0 to 4, are defined in ToBI, where *4* marks the end of an intonational phrase boundary, the most significant prosodic constituent boundary, and 3 marks an intermediate phrase boundary, the second most significant prosodic phrase boundary. In addition, 1 is the default boundary and 0 means no juncture between two adjacent words. Thus *break index* essentially encodes a hierarchical prosodic constituent structure. In addition to *break index*, *pitch accent (accent)* is associated with a significant excursion in a pitch contour. It often marks the lexical item with which it is associated as prominent. Both phrase accent (Pa) and boundary tone (Bt) control the shape of a pitch contour towards or at the end of an intonational or intermediate phrase.

***Acoustic features:*** They are encoded as pointers to a parametric segment database in which temporal sequences of vectors of parameters of speech segments are stored. Typical acoustic features encoded in the database include waveforms and parameters related to pitch, duration, and amplitude.

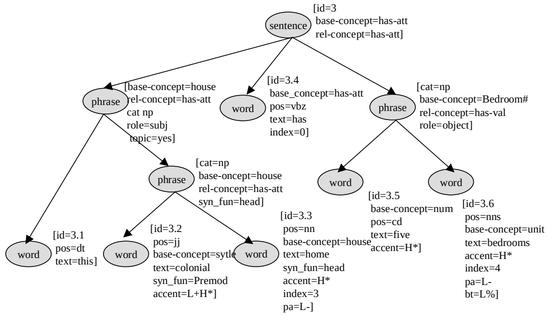


**Figure 3: A ReaTree**

Figure 3 shows an example of the ReaTree of *This colonial home has five bedrooms*. In this representation, there are four basic elements : *sentence, clause* (not in the example), *phras*e, and *word*. A *sentence* element is associated with a unique sentence id, pointing to the corresponding SemGraph. In addition, *phrase* is associated with features such as *base_concept*, *rel_concept*, *syntactic category, grammar role,* and *syntactic function.* In addition, it also associates with discourse features such as whether a phrase is the *topic* of a sentence. A *clause* is an embeded sentence. It associates with features similar to those of a phrase. Finally, the main features associated with a word include the text itself, the part-of-speech, the syntactic function, the associated ToBI prosodic features and an unique word id, pointing to the acoustic parameters represented in the speech segment database.

Finally, both the SemGraph and ReaTree are represented in XML because it is flexible enough to represent complicated structures, and at the same time, it also facilitates parsing and

[1] ToBI is a prosody annotation convention for American English.

searching that are essential for instance-based learning.

## 4 Using the corpus for generation

We now briefly describe how the annotated speech corpus can be used in CTS generation. Instance-based learning is lazy learning. It focuses on how to find similar instances in the training corpus and how to reconstruct new instances if a proper training instance is not found. There are three elements in the core of instance-based learning: similarity metrics, search algorithms, and reconstruction processes. Among them, the definitions of similarity metrics are based on the features annotated in the corpus. In addition, we employed a searching and matching algorithm that is also based on the structures of the annotated instances. Since the detailed descriptions of the similarity metrics as well as the searching and reconstruction algorithm are not the foci of this paper, we instead briefly describe how speech can be generated based on annotations in a SemGraph and ReaTree.

Our generation algorithm starts with a *diff* function that measures the difference (or similarity) between the SemGraph of a new input and those of corpus instances. To narrow down the search space, we focus on the top *n* matching corpus instances. If the result of *diff* for the top-matching training instance equals to zero, indicating an exact match, the entire matching instance is reused. In this case, SEGUE not only reuses the sentence structure and the wording but also the pronunciation, prosody, and waveforms. Thus, the resulting speech has high quality because the entire natural spoken utterance is reused. In general, for a domain-specific application, if a corpus is designed properly, there will be a significant number of cases falling in this category. However, if the result of *diff* is greater than zero, a set of revision operators are generated based on the difference. Typical revision operators include *remove, insert,* and *replace.* The *remove* operator deletes extra concepts or relations as well as their associated subtrees. The *insert* operator adds a new concept or relation. The *replace* operator only applies to *has-value* relations. It instantiates a variable with a different value. For example, if the input SemGraph is shown in Figure 5, and the closest matching training SemGraph is shown in Figure 2, the resulting *diff* operators will be:

1. *remove* has-att (House, HouseSize)
2. *remove* has-value (HouseSize, 9000)
3. *insert* has-att (House, HouseType)
4. *insert* has-value (HouseType, attached)
5. *replace* has-value (bedroom#, 4, 3)
6. *replace* has-value (bathroom#,3,2)

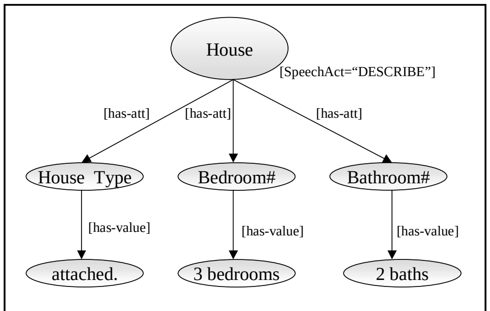


**Figure 5: An Input SemGraph**

Given a set of revision operators, the next step is to transform the corresponding ReaTree into one that can convey the meanings of the input SemGraph. In general, each operator is associated with a cost function. The overall cost function is a weighted combination of five subordinate cost functions: the *discourse cost, syntactic cost, lexical cost, prosodic cost,* and *acoustic cost*. All the cost functions measure the impact of applying an operator to a ReaTree. For example, *syntactic cost* measures how a revision operator affects the syntactic structure of a ReaTree. If an operator has little impact on the soundness of a syntactic structure, the *syntactic cost* will be low. In contrast, if applying an operator results in incomplete structures, the syntactic cost will be high. Similarly, in term of acoustic cost, if applying an operator, such as *insertion*, results in significant discontinuity between existing and new speech segments, the acoustic cost will be high.

In order to apply a *remove* operator to a ReaTree, the system first searches for a subtree that conveys the specified concepts/relation. Breaking a link on the subtree removes a concept or relation from the ReaTree. Breaking different links on the tree results in different *remove costs*. Similarly, when an *insert* operator is applied, the system first searches for a tree/subtree that communicates the specified relation, then it decides where and how to append the tree/subtree to a ReaTree. The difference in selecting a subtree as well as the difference in choosing a location to append the subtree may result in different *insert costs.* The *replace* operator searches for all the occurrences of a variable and replaces the existing value with a specified value. A *replace* operator is also associated with a *replace cost*. Depending on which occurrence of the word/phrase is used as the replacement, the *prosodic cost* and *acoustic cost* will be different, which in turn results in different *replace costs*. After applying all the operators, the lower the overall cost, the better the overall generation quality. After we repeat the entire process to convert the top *n* matching ReaTrees, the one with the lowest cost is the one to be generated by SEGUE. Our current prototype system only covers one type of speech act, *describe*, and a sub-domain of our application, *house descriptions.*

## 5 Related Work

Instance-based domain-specific speech synthesis is quite common (Donovan, 1997, Taylor, 2000). In contrast, most NLG systems use grammar-based approaches (Elhadad, 1993, Lavoie, 1997). Recently, machine learning-based NLG gains attentions (Ratnaparkhi, 2000, Walker, 2001, Oberlander, 2000, Varges, 2001, Langkilde, 2000). However, except for a few template-based systems (Yi, 1998), most CTS systems still use different platforms for NLG and speech synthesis. This uncoupled CTS architecture has inherent integration problems.

In terms of corpus design for CTS generation, until now, designing a single speech corpus for both NLG and speech synthesis in integrated CTS generation has not been systematically addressed. In (Theune, 2001), a speech corpus is designed only for TTS. No corpus is needed for its template-based NLG.

## 6 Conclusions

In this paper, we present a new uniform framework for systematically generating both text and speech using a single speech corpus. One of our research foci is on the design of a speech corpus for both text and speech generation. This framework facilitates the reuse of sentence structure, wording, prosody and speech waveforms simultaneously

## References

J. Bachenko and E. Fitzpatrick. *A computational grammar of discourse-neutral prosodic phrasing in English*. Computational Linguistics, 16(3): 155-170, 1990.

R. Donovan, M. Franz, J. Sorensen and S. Roukos. 1999. Phrase *Splicing and Variable Substitution Using the IBM Trainable Speech synthesis System.* Proceedings of ICASSP99. Phoenix, AZ.

Michael Elhadad. 1993. *Using Argumentation to Control Lexical Choice: A Functional Unification Implementation*. PhD Thesis. Columbia University.

I. Langkilde 2000. *Forest-Based Statistical Sentence Generation*. Proceedings of ANLP-NAACL00. 170-177, Seattle, WA.

B. Lavoie and O. Rambow. 1997. *A Fast and Portable Realizer for Text Generation Systems.* Proceedings of ANLP'97. Washington, DC.

J. Oberlander and C. Brew. 2000. *Stochastic text generation.* Philosophical Transactions of the Royal Society, Series A,(358) 1373--1385.

Shimei Pan. 2002. *Prosody Modeling in Concept-to-Speech Generation.* PhD thesis. Columbia University.

Adwait Ratnaparkhi. 2000. *Trainable Methods for Surface Natural Language Generation*. Proceedings of ANLP/NAACL'00. 194-201. Seattle, WA.

K. Silverman, M. Beckman, J. Petrelli, M. Ostendorf, C. Wightman, P. Price, J. Pierrehumbert, & J. Hirschberg. 1996. *ToBI: A standard for labeling English prosody*. Proceedings of ICSLP 92, (2) 867-870.

P. Taylor. 2000. *Concept-to-Speech by Phonological Structure Matching*. Philosophical Transactions of the Royal Society, Series A.

M. Theune and E. Klabbers. 2001. *From Data-to-Speech: A General Approach.* Natural Language Engineering. 7 (1): 47-86.

Sebastian Varges and Chris Mellish. 2001. *Instance_Based Natural Language Generation.* Proceedings of NAACL'01. Pittsburgh, PA.

Marilyn Walker and Owen Rambow. 2001 *SPoT: A Trainable Sentence Planner.* Proceedings of NAACL01. Pittsburgh, PA.

Hao Yan. 2000. *Paired Speech and Gesture Generation in Embodied Conversation Agents.* Master's thesis, MIT.

Jon Yi. 1998. *Natural-sounding speech synthesis using variable-length units*. Master's thesis, MIT.