Programming by Voice, VocalProgramming

Stephen C. Arnold
College of Information Science and Technology
3431 Chestnut St.
Philadelphia, PA 19104 USA
+01 215 895 6970
stephen.arnold@cis.drexel.edu

Leo Mark and John Goldthwaite[1]
College of Computing and
Center for Rehabilitation Technology
Georgia Institute of Technology
Atlanta, GA 30332

ABSTRACT
A system that enables a person to program without typing is needed because of the high incidents of repetitive stress injuries among people who program. This paper presents a design for a system that generates environments that enables people to program by voice and a method of determining if the system is successful. It also shows how this generator can be used to support entering data and writing XML documents.

Keywords
Computer programming, voice recognition, XML

Motivation

Work involving long periods of typing has been linked to an increased risk of Repetitive Stress Injury (RSI), also known as Repetitive Motion Injury or Cumulative Trauma Disorder. The problem is receiving growing attention as it affects more managers, professionals and white-collar workers. These injuries take the form of Carpal Tunnel Syndrome (CTS), tendonitis, severe neck pain or back pain. The total cost for RSI of all types is estimated at almost $6 billion annually.

Carpal Tunnel Syndrome results in pain and suffering for the individual and family. Occupational illnesses of this type result in loss of self-esteem, reduced quality of life and disruption of family life. None of these factors can be quantified or expressed in monetary terms but they add to the disease's enormous burden on society.

The symptoms of repetitive stress injuries appear gradually over months or years. Carpal Tunnel Syndrome usually starts with pain in the wrists at night or upon waking. Over time numbness or tingling will develop in the thumb and two fingers enervated by the medial nerve. If no intervention is taken, damage to the medial nerve will occur and the muscles of the thumb will begin to atrophy. A noticeable loss of grip strength will occur at some point. Finally, all sensation will be lost in the three digits and their muscles will be permanently damaged.

Prevention is a cheaper solution than attempting to repair the damage caused by CTS. CTS is believed to be caused by a combination of the repetitive, rapid motions of keystrokes over long periods of time exacerbated by downward extension of the wrists or continuous pressure on the wrists. A height adjustable office workstation with padded wrist rest and adjustable chair will facilitate typing in the correct posture and maybe sufficient to prevent injury for workers with a light workload. It is also recommended that workers shift tasks or take a break for five to ten minutes after each hour of typing. Unfortunately, many professionals such as newspaper reporters, computer programmers, authors and researchers work under tight deadlines and continue to work disregarding these recommendations. They should stop and shift to other tasks when fatigued and especially when pain or discomfort begins. If they must continue the task, they should shift to an alternate method of input such as voice recognition. [1]

Voice recognition systems have improved enormously through improvement in the software itself and from the vast increase in computational power over the last decade. In addition, the price of the voice recognition software has dropped from $7500 in 1987 to $100 today, making it an inexpensive accommodation for disabled workers. Current products such as Dragon Naturally Speaking, IBM Via Voice, and Lernout and Hausbie's Voice Xpress recognize continuous natural language speech at 20 to 60 words per minute with an error rate of about 2 to 5 percent. Newspaper reporters and office workers who need to continue type with a general vocabulary have

[1] Leo Mark 01 404 894 2746,
leomark@cc.gatech.edu
John Goldthwaite (+01 404 894 0563
john.goldthwaite@arch.gatech.edu

adopted these tools allowing them to work while recovering from CTS.

Dictation with the current generation of recognition software is much more difficult for programming tasks. Although keywords can be recognized, the majority of text in a program is variable and procedure names that are composites of words and abbreviations. These composites are not handled by recognition systems and must be spelled. Computer languages use punctuation symbols for equivalence, relational expressions, variable declaration and parameter handling in ways that are different syntactically from natural language. As a result an entirely different type of recognition system is needed for programming. We propose to build a generator for voice recognition syntax-directed programming environments. The generated programming environments will take advantage of the formal grammar of programming languages to simplify programming by voice.

The potential for this technology is not limited to programming languages. Extensible Markup Language (XML), the emerging standard for Web page definition and authoring, is based on document type definitions (DTD) or XML Schemas. An XML DTD or Schema is a grammatical definition of the contents of a class of XML documents containing a given, structure and representation. A particular XML DTD or XML Schema is a language that defines how well-formed XML documents of that type are written. We can use our voice generator technology to create a voice recognition syntax-directed environment for any XML DTD or XML Schema.

A major motivation for the design given here is to bundle the system for programming with popular commercially available voice recognition software. We expect this popular commercially available voice recognition engine recognition rates to improve over time. The improvements will be integrated into the voice recognition system for programming without significant programming efforts.

RELATED RESEARCH

The related research falls in two groups, research in programming languages and compilers that is fundamental to our work and voice recognition to support computer programming. Because XML DTDs and XML Schemas can also be considered as specialized programming language specification systems, these are also related.

Among the research results fundamental to the work proposed here are formal grammars, in particular context-free languages [7]; compiler-compilers, like YACC [8]; syntax-directed editor generators, like the Cornell Program Synthesizer [12]; grammars as a useful representation for data structures [5], [6].

John Edwards reports that the fundamental technology for continuous voice recognition significantly improved in 1997 [4].

Like Srinivasan and Vergo [11], our design is based on a commercially available voice recognition development environment. Their work focuses on making the development environment better. We are focusing on making a programming environment that can be controlled by voice alone.

Leopold and Amber [9] have developed a method of “writing” code for a Visual Programming Language that uses a combination of Voice, handwriting, and Vocal Programmer is significantly different than VoiceXML [13]. VoiceXML is XML language for writing voice enabled applications, not a general system for entering XML documents.

DESIGN

We have designed a generator for voice recognition syntax-directed programming environments.

This generator, VocalGenerator, takes as input a context-free grammar (CFG) for a programming language, such as Basic, C, C++, Java, or an XML DTD or XML Schema, together with a voice vocabulary for that language. The voice vocabulary includes the literals from the programming language; the names of classes and functions from the class and function libraries available to the programmer; and a list of class, function, and variable names specific to the program file. These are associated with a list of pronounceable words and phrases that the programmer would use to enter and edit a program in the language.

VocalGenerator generates as output a programming environment in which the programmer can write programs by voice input alone. This programming environment includes a voice recognition syntax-directed program editor. This voice recognition syntax-directed program editor aids the programmer by providing automatic completion of program text and appropriate navigation relative to the specific programming language. A voice recognition syntax-directed program editor can also be used to edit programs that have already been developed. It analyzes a given program and generate the vocabulary used in the specific program.

We refer to the generated voice recognition syntax-directed programming environments with names, such as VocalBasic, VocalC, VocalC++, or VocalJava.

Once VocalGenerator has been built it can be used several times to generate programming environments that support voice recognition syntax-directed programming in specific programming languages. Relatively little effort is required to prepare the input grammar and voice vocabulary for each additional programming language.

A voice recognition syntax-directed program editor makes it easier to write a program by voice than a standard text editor. Standard voice recognition text editors are not effective in entering and editing computer programs. Stephen C. Arnold (an author of this paper) has personally tried programming in a voice recognition editor intended for standard text input after developing bilateral carpal tunnel syndrome and was not able to write even small programs.

Syntax-directed editors provide two major capabilities that standard text editors do not: navigation and selection of sections of code based on the structure of the program and automatic completion of program constructs. A syntax-directed editor, like VocalJava, makes selecting a whole class or program block a simple operation because it recognizes these as structures within the program text. As an example, a programmer could ask to move up two classes and to the first block within the class definition and delete that block. A syntax-directed editor, like VocalJava, would also be able to complete statements based on the beginning of the statement. For example, the word "if" would result in the completion of the structure for an if-statement consisting of "if", an "(", a space for the condition, an ")", an "{", a space for a block of statements, and an "}".

A voice recognition syntax-directed editor has an additional advantage over a voice recognition text editor because of its limited vocabulary. Programming languages have a very small vocabulary compared to natural languages. The voice recognition engine will have fewer words to recognize, so it will have a better chance of getting the word right.

Figure 1 illustrates the use of a voice interface for syntax-directed programming. It shows the first eight steps in the development of a simple program. In each step, the arrow shows the cursor position at the beginning of the step and the bubble shows the voice command(s).

Step 1: The cursor is pointing at the non-terminal, <program>. The command "begin" triggers the production ,<program> ::= BEGIN <statement-seq> END;

Step 2: The cursor is pointing at "BEGIN". The commands, "move down, statement-seq", moves the cursor to the next non-terminal, <statement-seq>, and triggers the production, <statement-seq> ::= <statement>; <statement-seq>

Step 3: The cursor is pointing at the non-terminsl, <statement>. The command, "<statement>, prepares for the entry of a statement.

Step 4: The cursor is pointing at the non-terminal, <statement>. The command, "if", triggers the production, <statement> ::= IF (<condition>) THEN <statement> ELSE <statement>.

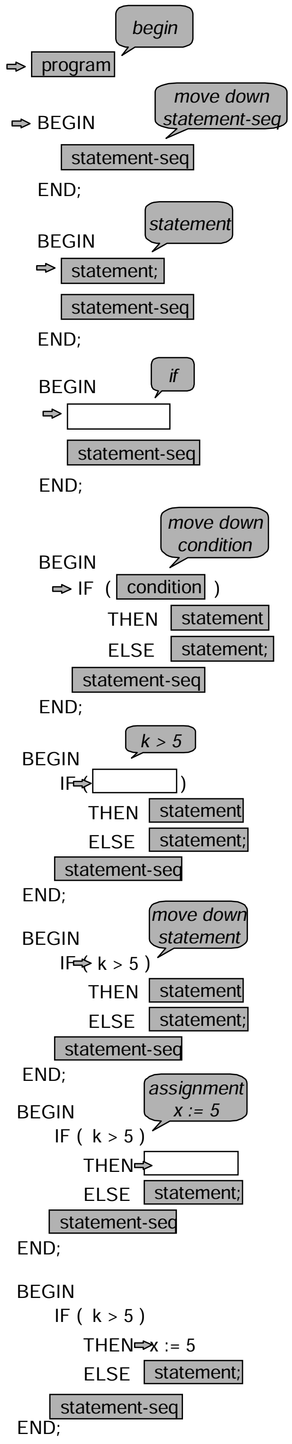

Figure 1: Example of the use of a voice interface for syntax-directed programming

Step 5:: The cursor is pointing at "IF". The commands, "move down, condition", moves the cursor to the next non-terminal, <condition>, and prepares for the entry of a condition.

Step 6: The cursor is pointing at the non-terminal, <condition>. The command, "k>5", enters that condition into the program.

Step 7: The cursor is pointing at the non-terminal, <condition>, just entered. The commands, "move down, statement", moves the cursor to the first non-terminal, <statement>, and prepares ifor the entry of a statement.

Step 8: The cursor is pointing at the non-terminal, <statement>. The commands, "assignment, x:=5", enter that assignment statement into the program.

Step 9: The cursor is pointing at the assignment statement just entered.

Applications extend beyond computer programming to data entry and editing. A data entry and editing form has a structure that can be defined by a grammar. *Figure 2* illustrates a grammar for a data entry and editing form.

```
CENSUS     ::= RESIDENCE*
RESIDENCE ::= PERSON*, ADDRESS
PERSON     ::= NAME, SEX
NAME        ::= FIRST-NAME, LAST-NAME
SEX          ::= MALE | FEMALE
ADDRESS    ::= STREET, CITY, STATE, ZIP
STREET      ::= NUMBER, STREET-NAME
```

Figure 2: Grammar for a data entry and editing form for the census

Once this grammar and a vocabulary for it have been defined, VocalGenerator will generate the voice interface supporting data entry and editing through the form the same way it generated a programming interface for a programming language. The data entry and editing form generated by VocalGenerator on the grammar in *Figure 2* is illustrated in *Figure 3*.

The example form in *Figure 3* shows how we plan to deal with each one of the basic constructs in a CFG when generating the form interface:

- *Sequence*: Five non-terminals, RESIDENCE, PERSON, NAME, ADDRESS and STREET, are sequences. Each one is the root in a tree with the right-hand side non-terminals or terminals as branches.
- *Iteration:* Two non-terminals, PERSON and RESIDENCE, can be iterated. Additional data entry fields for PERSON and RESIDENCE can be generated by clicking the "more?" buttons.
- *Choice:* One non-terminal, SEX, is implemented as a "radio" button.

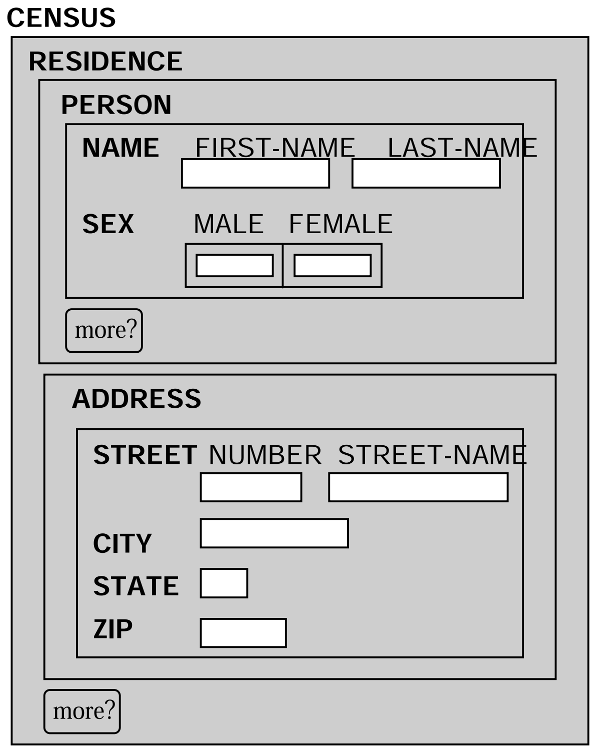

Figure 3: Data entry and editing form generated from the grammar in Figure 2

One of the challenging problems in this research is dealing with multiple levels of nesting and with in-line use of nesting and iteration. One solution to the problem is to "normalize" or "denormalize" the grammar. However, the grammar must make sense to the user of the system. Users should not have to understand CFGs, but must be able to understand the relationships between different parts of the language they are programming in or form the are entering data into.

A number of different types of commands are needed in the syntax-directed programming interface as well as in the forms-based data entry and editing interface. They include commands for

- *operating system:* file management, compilation, execution, debugging, etc.
- *moving around:* scroll-up, scroll-down, page-up, page-down, top, bottom, left, right, etc.
- *typing:* characters, such as "(", ")", "<", ">", "=", "+", "-", "*", "/". Erase, delete and tab. Shift and return.
- *editing:* ctrl, esc, highlight, cut, copy, paste, find, and replace.
- *programming:* vocabulary for a programming language, such as "integer", "string", "boolean", "begin", "end", "if-then-else", "while", "for", "repeat", "case", "procedure", "function" and "comment". Variable names and constants. Syntax-directed motion, such as "next

statement", "previous statement", "next non-terminal", "previous non-terminal", etc.

Fortunately, with the exception of commands for programming, most of these types of commands are provided for by voice recognition software, and we shall concentrate on commands for programming, only.

FUTURE POTENTIAL

The potential users of this type of technology is not limited to computer programmers. A new extensible markup language, XML, will fast become a standard for information interchange on the Internet. The use of XML will be pervasive in eCommerce and on the Web in general within a very short period of time. XML is particularly interesting from the point-of-view of this project because of the concept of an XML Schema. An XML Schema is a grammatical definition of a collection of XML documents containing data with the same meaning, structure and representation. The same way a grammar for a programming language defines all well-formed and valid programs that can be written in the language defined by the grammar, an XML Schema defines all well-formed and valid XML documents that can be written in the language defined by the XML Schema. With some modifications, VocalGenerator can be cloned to understand not only BNF definitions of CFGs, but also XML Schema definitions.

EVALUATION OF VOCAL PROGRAMMER

Once we have built the VocalGenerator, we shall test the following two hypotheses:

- Hypothesis 1: Programming is faster using a syntax directed voice recognition system than using a natural language recognition system.
- Hypothesis 2: A programmer with Carpal Tunnel Syndrome or other motor impairment will be able to program at a rate that is fast enough to maintain competitive employment. Typing speed is not the limiting factor for employment unless the rate is less than critical minimum number of lines per day.

To test these hypotheses we will build VocalProgrammer with log capabilities that will capture vocal commands as interpreted by the voice engine and the voice engines text output. Each speech command will be logged and time stamped for later analysis. We will also create a modified keyboard handler to capture typing outside of the VocalProgrammer. The keyboard logger will be used to record the typing of both disabled and non-disabled experimental subjects. VocalProgrammers log will record the voice command, the text entered in the editor, and a time stamp. The keyboard log will record each keystroke, the destination application, and a time stamp.

We will test this system on students with severe mobility impairment such as quadriplegia, carpal tunnel syndrome, cerebral palsy, or rheumatoid arthritis that restricts the ability to type. Some of these individuals will likely already use voice recognition as their primary input method. We will also recruit a sample of ten matching non-disabled programmers to develop baseline data about typing rate for individuals without disabilities.

We will provide Dragon Naturally Speaking 4.0 (DNS) for the ten individuals with disabilities. We will give each programmer a two-day tutorial on the use of voice recognition software to insure that each is fully trained in its use. We will follow their use of DNS voice recognition software for a period of six months, providing additional tutoring as needed. The six-month training period will insure that all subjects are proficient in the use of voice recognition.

After six months, the programmers will be divided into two groups. Half will continue to use Dragon Naturally Speaking, the other half will use Dragon with our syntax-directed editor. We will monitor their programming for a period of one month, logging all input. We will also monitor the matching programmers without disabilities.

To analyze the log data, we will develop a visualization program that will convert the log into a timeline. The timeline will display tracks for each program in use. The users typing will be displayed with a graph of the moving average of the typing rate. It will measure the typing rate for each line as well as for user selected segments of the timeline.

Using this tool, we will be able to analyze input for bursts of typing that are likely to occur in programming. It will also permit a finer analysis of verbal programming so that we can detect and document error patterns, error correction strategies as well as input and editing strategies employed by the subjects.

Analysis of the non-disabled programmers work will give us input rates for various tasks. We will be able to compare these rates to those of the programmers using Dragon Naturally Speaking and our syntax directed voice system. We will determine whether the programmers are working at comparable rates and whether there is a significant difference between the three groups.

ENABLING TECHNOLOGIES

VocalGenerator combines two existing computer technologies, syntax-directed editor generators and voice recognition development environments. To the best of our knowledge, such a combination does not currently exist.

Voice recognition software has become practical to use on personal computers. It can easily run on personal computers and is often integrated into existing software development environments, such as Visual Studio.

Voice recognition software has been built to support text entry. Though such systems are still difficult to use, they are effective. The existing voice recognition software is not effective for programming. The differences between natural language and computer languages is significant enough that attempting to use voice recognition software for programming quickly becomes so slow and frustrating that it makes it impractical.

Voice recognition development environments are comer-cially available. These environments make it possible to develop voice recognition applications without having to develop the voice recognition system itself. It is only a matter of using the existing voice recognition engine in the software system the developer decides to create.

Dragon NaturallySpeaking ActiveX Components is a commercially available voice recognition engine. It is the engine of the highly rated Dragon NaturallySpeaking voice recognition system. It is quite capable of providing the voice recognition capabilities necessary for building a voice recognition system to support programming. The Dragon NaturallySpeaking ActiveX Components contains software for building a library of vocabulary for a computer language. This library consists of the words that are used in writing a program. These words are the tokens of the programming language, the names of functions and procedures found in standard libraries, and the names of variables, functions, and procedures that exist in the specific program being written.

Syntax-directed editors have been a practical technology for almost 20 years. Syntax-directed editors take advantage of the highly regular structure of computer languages. Instead of requiring a programmer to type every character of a statement, the editor determines the remainder of the statement based on its beginning.

Since syntax-directed editors use a formal description of the computer language, once the system has been built for one computer language it will be very easy to modify to make it work for a different computer language. So, once a syntax-directed editor is built for a specific computer language, such as Java, it is a relatively easy task to make it work with another language, such as C++.

The Cornell Program Synthesizer is a syntax-directed editor capable of being customized to support specific computer languages. The technology is well understood and easily reproduced if necessary. It is quite capable of providing the syntax-directed editing needed to build a voice recognition system for supporting programming.

After VocalGenerator has been built for one programming language, only two things would be required for a different programming language: a new language specification and a new vocabulary library. The language specification should be very simple to find and would require only minor modifications from one either commercially available or available in public the domain. The vocabulary library for the new language should also be relatively easy to build. A list of names of procedures and functions will need to be obtained. A person would have to pronounce each of these names to the system.

It is clear that the technology needed to build Vocal Generator is readily available.

HOW WE WILL BUILD IT

VocalProgrammer will be constructed using two commercially available software packages: the Dragon NaturallySpeaking development tools and Microsoft Visual C++. VocalProgrammer will be made available as freeware. It will require the user to have purchased Dragon NaturallySpeaking Professional Version, because it requires the speech recognition engine within the Dragon NaturallySpeaking Professional Version. The investigators will only keep sufficient rights to the software system so that they can continue research on it.

From a design perspective, VocalProgrammer is a combination of two technologies: voice recognition and syntax-directed editor generators.

The voice recognition capabilities needed to produce VocalProgrammer are completely within the software of the Dragon NaturallySpeaking development tools. The programming environment generated for a specific pro-gramming language by VocalProgrammer will be able to be fully controlled by voice. The Dragon Naturally Speaking SDK contains a library of calls that provide the ability to control a menu system by voice and enter text continuously [1]. The programming environment will be constructed such that all the commands of the environment can be executed from menus. The code that will be entered by the users will be entered using the continuous speech mode supported by the Dragon NaturallySpeaking SDK. The Dragon NaturallySpeaking Enhanced Runtime [2], that is contained within the Dragon NaturallySpeaking Professional Version, is capable of supporting a completely application defined vocabulary. This will enable VocalProgrammer to replace the default natural language vocabulary of Dragon NaturallySpeaking with the vocabulary of the specific programming language.

The word set for each specific programming language can be extracted directly out of the documentation of that language. We are targeting Java because its full documentation set is available for free on-line. This word set will be turned into a vocabulary recognized by the Dragon NaturallySpeaking Extended Runtime by the Dragon NaturalVoc Tool [3]. This tool also creates statistical relationships between the words in the vocabulary produced by analyzing large quantities of text. This text can be generated from the public domain Java source code. This will produce a vocabulary for VocalJava.

Syntax directed editor generators are a well-understood technology [12]. Syntax directed editor generators are constructed from two technologies: editors and parser generators. The functionality needed to construct an editor readily exists within Microsoft Visual C++. The source code for a number of parser generators exist in the public domain in C++ [10]. This source code can easily be ported to Microsoft Visual C++ if it is not already in a form that can be compiled by this system.

The core of the software development effort involved in building VocalProgrammer is integrating existing software systems.

These are not the only issues in the construction of VocalProgrammer. Computer languages are not spoken, they exist only in written form. A significant effort will be necessary to determine how to vocalize these languages. This is an issue that has not been explored yet and may have applications beyond programming by voice. When programmers discuss their code in situations, such as code reviews, a standard way of speaking the code itself may be of value. In this way, this research may have value beyond its original objectives of helping disabled computer programmers program again.

REFERENCES

[1] Dragon Systems, *Dragon NaturallySpeaking SDK, C++ and SAPI Guide and Reference*, Newton, Mass. 1999.

[2] Dragon Systems, Dragon NaturallySpeaking SDK, *Integrating a Runtime into Your Speech-Enabled Application*, Newton, Mass. 1999.

[3] Dragon Systems, *Dragon NaturallySpeaking NaturalVoc Tools, User's Guide*, Newton, Mass. 1999.

[4] John Edwards, Voice Based Interfaces make better PC Listeners, *Computer*, August 1997.

[5] G.H. Gonnet and F. Tompa. A constructive approach to the design of algorithms and their data structures. *Communications of the ACM*, Vol. 26, No. 11, pp. 912-920, 1983.

[6] M. Gyssens, J. Paredaens, and D. VanGucht. A grammar-based approach towards unifying hierarchical data models (extended abstract). In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Portland, Oregon, 1989.

[7] J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1979.

[8] S.C. Johnson. YACC: Yet another compiler-compiler. Technical report, Bell Labs., 1978.

[9] J. Leopold and A. Amber, Keyboardless Visual Programming Using Voice, *1997 IEEE Symposium on Visual Languages (VL '97)* Isle of Capri, ITALY September 23-26, 1997.

[10] David Muir Sharnoff, Catolog of Compilers: BNF, http://www.idiom.com/free-compilers/LANG/BNF-1.html, downloaded on Feb. 4, 2000.

[11] Savitha Srinivasan and John Vergo, Object Oriented Reuse: Experience in Developing a Framework for Speech Recognition Applications, *The 20th International Conference on Software Engineering*, Kyoto, Japan, pp 322-330, 19 - 25 April 1998.

[12] T. Teitelbaum and T. Reps. The Cornell Program Synthesizer: A Syntax-Directed Programming Environment. *Communications of the ACM,* Vol. 24, No. 9, pp. 563-573, 1981.

[13] Voice XML Forum 2000, Voice XML Forum, Voice eXtensible Markup Language, http://www.voicexml.org/specs/VoiceXML-100.pdf, downloaded on May 8, 2000