Building Domain Specific Languages for Voice Recognition Applications

Cristian IONIȚĂ
Academy of Economic Studies, Bucharest
Cristian.Ionita@softmentor.ro

This paper presents a method of implementing the voice recognition for the control of software applications. The solutions proposed are based on transforming a subset of the natural language in commands recognized by the application using a formal language defined by the means of a context free grammar. At the end of the paper is presented the modality of integration of voice recognition and of voice synthesis for the Romanian language in Windows applications.
***Keywords**: voice recognition, formal languages, context free grammars, text to speech.*

Introduction

Large scale usage of software products imposed a significant development and diversification of the human – computer interaction methods. A special category of interfaces is that based on voice recognition and synthesis. Using this type of interface is necessary for voice recognition and synthesis. It also helps the persons with disabilities to use the applications and it is useful in the cases when the classical interaction access is not possible or is dangerous (equipment manipulation, medical interventions, etc.).

Recognizing the natural language is a very difficult task for a computer. The main difficulties in the process of recognition are ([3]):

- unlike people, the voice recognition algorithms do not have the general knowledge necessary for interpreting and predicting the words;
- impossibility of the voice recognition programs to receive and interpret the non-verbal communication;
- the lacks of the intervals between words and the omissions frequent in the current speech;
- the large variability of the speech; the pronunciation of the words differ according to the speaker's characteristics (style, sex, anatomy of the vocal apparatus and dialect), but also according to context (the speed of speech);
- identification and elimination of background noise.

Some of these difficulties can be diminished by limiting the flexibility of the voice recognition module. Limiting the vocabulary and the word order that can be used for the control of the application can lead to a significant increase of the accuracy and speed of the recognition process. In the following lines is presented a modality of building the applications that use voice synthesis and recognition for Romanian language based on free context grammars.

1. Grammars for voice recognition

The languages used for communication can be classified in natural and formal languages. Natural languages like the spoken languages (Romanian, English, French), music and sculpture are too complex to be completely processed by the means of algorithms. The formal languages are languages projected for an automatic processing and described through a formal grammar. A grammar is an exact, finite and complete description of a formal language.

A grammar is made up of a set of four components ([4]):

- a set of terminal symbols (basic words of the language);
- a set of non-terminal symbols (symbols built on the basis of terminal symbols);
- a finite set of production rules (that specify the ways the non-terminal symbols can be built);
- a start symbol.

A simple way to represent a formal grammar is Backus-Naur form (BNF, [2]). It uses pro-

duction rules of type *A*:== *B*, where *A* is a un non-terminal symbol and *B* is an expression that can contain terminal and non-terminal symbols. The definition of B can contain the following types of elements:
- sequences: $B_1\ B_2\ ...B_n$;
- alternatives: $B_1 \mid B_2 \mid ... \mid B_n$;
- brackets for combining sequences and alternatives;
- *?* symbols for indicating an optional element, * for indicating repetitions of 0 or more elements and + for indicating repetitions of 1 or more elements.

For exemplification we will take into account a language for describing a list of colors. A phrase in this language will consists of a series such as: $color_1$, $color_2$, $color_3$, ... $color_{n-1}$ and $color_n$. The BNF grammar corresponding to this language is:
COLOR :== "white" | "yellow" | "orange" | "red" | "blue" | "green" | "indigo "
SERIES _ COLORS:== COLOR (","COLOR)*
LIST_COLORS:== SERIES _ COLORS "and" COLOR
This grammar is formed of the terminal symbols *white, yellow, orange, red, blue, green, indigo,* and the non-terminal symbols *COLOR, SERIES_ COLORS, LIST_* COLORS, the three production rules corresponding to them and the start symbol *LIST_ COLORS*. An example of correct phrase in this language is: “green, yellow and indigo”. Each phrase of a formal language can be graphically represented as a tree according to the grammar associated to the language. Figure 1 presents the tree associated to the above mentioned phrase.

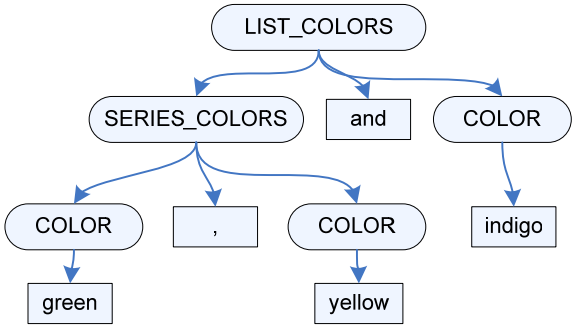

Fig.1. The arborescent representation of a phrase based on grammar

In the context of voice recognition, o grammar is a structured list of rules that identify phrases and words that can be used for voice recognition within an application. These rules offer to the application indications usable for a better recognition of the received sounds. Indications provided by grammar are used for the restriction of the selection base for words and increase the process accuracy.
The formal grammars were classified by Noam Chomsky ([2]) according to their descriptive force. Within the systems of voice recognition are used especially free context grammars (type 2). These are grammars that contain production rules *A :== expression*, where *A* is a non-terminal symbol and *expression* is a succession of terminal and non-terminal symbols. These are the languages that can be efficiently processed by a finite machine.
For describing the grammars used in voice recognition, World Wide Web Consortium (W3C) created the standard *Speech Recognition Grammar Specification* (SRGS, [8]). This standard was adopted on a large scale within the voice recognition solutions. The standard allows the complete description of the free context grammars used in voice recognition using the syntax XML or ABNF (*Augmented BNF*).
A SRGS grammar allows the programmer to indicate to the voice recognition module the following:
- the valid words;
- the word order and the way of composition;
- the language used and the pronunciation for each word;
- the semantic information associated to the language constructions.

The SRGS grammars are made up of a list of production rules. Each production rule is composed of terminal symbols, references to other rules, sequences, alternative structures, repetitive structures and semantic elements.
The syntax used for defining a rule is: *<rule id = "string" scope = ("public" | "private")> ... </rule>*. Identifiers associated to the rules can be used for referring to that rule within the grammar or from an external context (for public rules). Within a production rule can be

included terminal symbols (defined through <*item*> or direct inclusion within the rule) and references to other non-terminal symbols (defined through <*ruleref*>). The elements existent within a rule are treated implicitly as a part of a sequential structure. The alternative structure is introduced through the element <*one-of*>. The operators *?*, * and + of the BNF notation are stimulated through the attribute *repeat* applied top the symbols of the grammar.

The SRGS grammars can contain also semantic information for the automatic interpretation of the production rules ([9]). This information is included in the elements type <*tag*>.

2. Building a language for voice recognition

Using the voice recognition within an application involves defining the language of commands of the associated SRGS grammar. Developing such a language involves more steps of design and implementation.

The first in designing the language is the definition of the types of commands that should be supported by the application. For exemplification we will analyze an application that involves guessing a number by the user. The commands available in this case can be: “try number N” and “stop the game”.

After identifying the commands, they should be decomposed in the elements. The components identified together with the start symbol will become the public rules of the grammar. For this example, these rules will be:

```
<grammar xml:lang="EN-US" tag-
format="semantics-ms/1.0" version="1.0"
mode="voice"

xmlns="http://www.w3.org/2001/06/grammar
">

  <!-- Symbolul de start al gramaticii--
>
  <rule id="comanda" scope="public">
    <one-of>
      <item>
             <ruleref uri="#incercare"/>
       </item>
      <item>
             <ruleref uri="#oprire"/>
       </item>
    </one-of>
  </rule>

  <rule id="oprire" scope="public">
    <item>oprire joc</item>
  </rule>

  <rule id="incercare" scope="public">
    <item>incearca</item>
    <ruleref uri="#numar" />
  </rule>
</grammar>
```

In the next step should be identified the non-terminal symbols in the process of establishing the basic rules. For the analyzed example we have to explain the manner of defining a number. The simplified grammar for the recognition of the numbers in the interval 0 – 99 is:

```
  <rule id="numar">
    <one-of>
      <item>
             <ruleref uri="#cifra" />
       </item>
      <item>
             <ruleref uri="#numar_10_19"
       />
       </item>
      <item>
             <ruleref
       uri="#numar_20_100" />
       </item>
    </one-of>
  </rule>

  <rule id="numar_20_100">
    <ruleref uri="#cifra"/>
    <item>zeci</item>
    <item repeat="0-1">si</item>
    <item repeat="0-1">
      <ruleref uri="#cifra"/>
    </item>
  </rule>

  <rule id="numar_10_19">
    <one-of>
      <item>zece</item>
      <item>unsprezece</item>
      ...
    </one-of>
  </rule>

  <rule id="cifra">
    <one-of>
      <item>zero</item>
      <item>unu</item>
      <item>doi</item>
      ...
    </one-of>
</rule>
```

After establishing the basic rules and explaining all the non-terminal symbols, the next step is the process of refining the grammar for improving the quality of recognition. To this aim there will be taken into account the alternative forms of the commands or

adding information regarding the pronunciation of the words or the language used. For example, for the stop command we can take into account the following alternatives:

```
  <rule id="oprire" scope="public">
    <one-of>
      <item>
        <one-of>
          <item>oprire</item>
          <item>părăsire</item>
        </one-of>
        <item repeat="0-1">joc</item>
      </item>
      <item>
        închide
        <item repeat="0-
1">aplicaţia</item>
      </item>
    </one-of>
  </rule>
```

After this step, the process of building the grammar for commands recognition is completed. The engine for voice recognition is able to recognize the voice of the user and to produce the text in the specified language.

If the resulted text requires further processing to be used by the application, the process of designing can continue with the addition of new semantic information for the post processing of the resulted text. In the example above it should be useful to replace the text associated to the recognized numbers with the numeric representation. For associating a numeric property to a figure it will be used:

```
<rule id="cifra">
  <one-of>
    <item>
      zero
      <tag>$.valoareNumar={};
$.valoareNumar._value = 0;</tag>
    </item>
      …
  </one-of>
</rule>
```

The same way the production rules can be completed with the ECMA instructions for generating the numeric values corresponding to the other used numbers.

3. Integration of voice recognition in applications

Integrating the voice recognition and synthesis in applications that works on the Microsoft Windows platform can be done by using Microsoft Speech API library (SAPI, [6]). Windows Vista includes SAPI 5.3 version that implements the support for the standards SRGS and SSML ([8], [10]), as well as the support for the semantic interpretation using the ECMA language. SAPI can be used directly by the native applications or through *System.Speech* ([7]) library by the applications developed using platform .NET. Figure 2 presents the general architecture of a .NET application that uses SAPI ([1]).

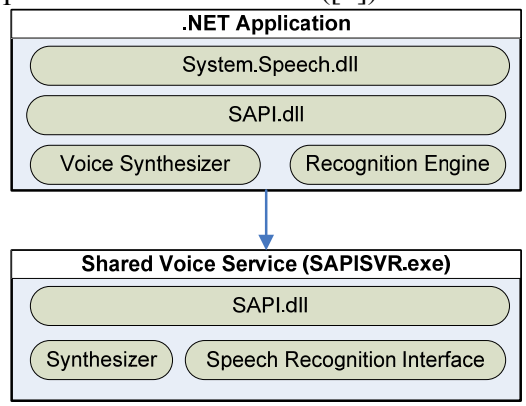

Fig.2. SAPI 5.3 general architecture (Windows Vista*)*

Building an application that uses voice recognition for taking over the commands from the users involves three basic steps: describing the commands supported by the application and using a SRGS grammar, starting the voice recognition engine and interpreting the results.

The grammar corresponding to the language can be built dynamically using the classes offered by .NET platform in the library *System.Speech.Recognition.SrgsGrammar* or using a XML folder compatible with SRGS. The dynamic building of the grammar is done this way:

```
// SRGS document creation
SrgsDocument gramatica = new
SrgsDocument();

// Grammar rule creation
SrgsRule regulaCifra = new
SrgsRule("Cifra");
SrgsOneOf alternativaCifra = new
SrgsOneOf(
    new SrgsItem("zero"),
    new SrgsItem("unu"), …);
regulaCifra.Add(alternativaCifra);

// We attach the rule to the
existing grammar
// and specify the start symbol
gramatica.Rules.Add(regulaCifra);
```

```
gramatica.Root = regulaCifra;
```

Using this mechanism any grammar can be built dynamically according to the SRGS standard. Initializing the voice recognition engine involves at least setting the used language, the entrance device, the grammar and the processing function:

```
// Speech recognition engine
creation
SpeechRecognitionEngine
motorRecunoaștere =
    new SpeechRecognitionEngine(new
CultureInfo("en-US"));

// We set the audio source
(microphone by default)
motorRecunoaștere.SetInputToDefaultA
udioDevice();

// We load the grammar into the
engine
motorRecunoaștere.LoadGrammar(new
Grammar(gramatica));
// We attach the handlers for result
procesing
motorRecunoaștere.SpeechRecognized +=
new
EventHandler<SpeechRecognizedEventArgs
> (motorRecunoaștere_SpeechRecognized);
// We start the engine in async mode
motorRecunoaștere.RecognizeAsync(Rec
ognizeMode.Multiple);
```

The search engine allows configuring also other parameters by the means of properties or of objects type *RecognizerInfo*. There can also be added functions for taking over the partial results of the voice synthesis: detecting the language, taking over the recognition alternatives or the audio data for alternative analyses in the case when a command could not be identified.

The results of voice recognition are transmitted in asynchrony to the application through the specified function at initializing the recognition engine. The data resulted from recognition are encapsulated in an object type *RecognitionResult*. This object contains the recognized text, the value obtained from the application of the semantic corresponding to the rules, the registered sound sequence and information referring to the recognition (the reliability, possible alternatives, etc.). The application can use this information to select the commands.

SAPI library allows also using the voice synthesis for communicating information to the user within a multimodal application. The synthesis engine can generate the sounds corresponding to the message received as a text or SSML document. The voice synthesis for messages in Romanian can be done using the *Carmen* voice module produced by *IVO Software* ([5]) company.

Conclusions

The voice recognition techniques based on restriction used by free context grammars presented within the paper can be utilized for building any voice controlled application by the means of a set of commands. The restriction of the vocabulary through the grammar leads to a recognition rate clearly higher than in the case of an unsupervised recognition.

References

[1] Brown R., *Exploring New Speech Recognition And Synthesis APIs In Windows Vista*, MSDN Magazine, January 2006

[2] Chomsky N., *On certain formal properties of grammars*, Information and Control, 1 (1959), pages 91-112

[3] Forsberg M., *Why is Speech Recognition Difficult?*, Chalmers University of Technology, Göteborg 2003

[4] Ioniţă C., "A Domain Specific Language for Secure Document Management", Proceedings of the Seventh International Conference on Informatics in Economy, ASE, Bucharest 2007

[5]IVOSoftware,http://www.ivosoftware.com/products/ivona_professional.html, IVONA Professional Presentation

[6]Microsoft, http://msdn2.microsoft.com/en-us/library/ms723627.aspx, Microsoft Speech API 5.3 Documentation

[7]Microsoft, http://msdn2.microsoft.com/en-us/library/system.speech.recognition.aspx, System.Speech.Recognition Namespace Documentation

[8] World Wide Web Consortium, *Speech Recognition Grammar Specification Version 1.0*, W3C Recommendation 2004

[9] World Wide Web Consortium, *Semantic Interpretation for Speech Recognition (SISR)* Version 1.0, W3C Recommendation 2007

[10] World Wide Web Consortium, *Speech Synthesis Markup Language (SSML) Version 1.0*, W3C Recommendation 2004