LPA-Speech: An Interface Between LPA-Prolog and Microsoft SAPI

Arlo Lyle
Artificial Intelligence Center
The University of Georgia
http://www.ai.uga.edu

February 8, 2006

1 Introduction

This paper describes the implementation and use of LPA-Speech, an interface between LPA-Prolog and Microsoft SAPI (Speech Application Programming Interface). LPA-Speech is an extension of SWI-Speech created by Jonathan McClain for PRONTO (Prolog Natural Language Toolkit), a package created by the Artificial Intelligence Center at the University of Georgia. It is fully compatible with LPA-Prolog version 4.11 and later and Microsoft SAPI version 5.1. This paper is divided into discussion of how to use LPA-Speech as well as the implementation of LPA-Speech.

1.1 Requirements

To use LPA-Speech it is necessary to have LPA-Prolog version 4.11 or later and Microsoft SAPI version 5.1 installed. For speech recognition, a functioning recognition profile is necessary. Recognition profiles are maintained under Speech in the Control Panel.

2 Using LPA-Speech

The following section describes how to use LPA-Speech within LPA-Prolog applications. The LPA-Speech package consists of three files, *LPA_Speech.pl*, *lpa_speech.dll* and *registry.dll*. These files must be contained in the same directory while being used. To begin using LPA-Speech, load *LPA_Speech.pl* into Prolog as follows:

```
:- ensure_loaded('LPA_Speech.pl').
```

LPA_Speech.pl will handle the process of loading and controlling the C++ functions contained in *lpa_speech.dll*.

2.1 Speech Recognition

LPA_Speech.pl allows for the control of speech recognition using the predicate `listen/1`. `listen/1` accesses SAPIs speech recogntion engine via `lpa_speech.dll`, listens for a line of speech input, and returns the input in the form of a compact string. An example of a query to `listen/1` is included below:

```
?- listen(Sentence).

Sentence = `Hello`
```

2.2 Speech Synthesis

LPA-Speech provides the user with two predicates for the control of speech synthesis, `speak/1` and `speak_with_attributes/1`. These predicates differ by the amount of control over the speech output they provide. `speak/1` uses the default voice settings of SAPI, giving the user no control over how the final output sounds. Both predicates allow input to be in the form of a character list, compact string, or a quoted atom. Thus, the following three queries speak "Hello world" with the default voice settings using `speak/1`:

```
?- speak("Hello world").

?- speak(`Hello world`).

?- speak('Hello world').
```

On the other hand, `speak_with_attributes/1` allows the user to set the voice qualities desired for a particular application. Because the likliehood of two different machines having the same voices installed on them is very small, SAPI does not allow for the specification of the exact voice to be used for speech output. Instead, it gives users control over voice output through the use of attributes. LPA-Speech provides control over four different attributes related to voice output, volume, rate of speech, age, and gender.

The volume of speech output is controlled using the predicate `set_volume/1`. The range of values for the volume attribute is 0 to 100, where 0 is the quietest setting and 100 is the loudest setting. Similarly, the rate of speech is controlled by the predicate `set_rate/1`. The range of values for the rate of speech is -10 to 10, where -10 is the slowest setting, 10 is the fastest setting, and 0 is average. The following example speaks "Hello world" at an average rate of speech and at full volume:

```
set_rate(0),
set_volume(100),
speak_with_attributes("Hello world").
```

The age and gender attributes are controlled by the predicates `set_age(+Age,+Condition)` and `get_gender(+Gender,+Condition)`. The settings of these attributes are used to search through the Windows Registry for an appropriate voice. For both of the above predicates, the condition value can be set to either optional or required. If the condition value is set to required the search will only return voices that match the required attribute. If no voices match the required attributes, then the default voice is returned. On the other hand, if the condition value is set to optional the search will return voices that do not match the optional attributes. However, voices that match the optional attributes are preferred over those that do not. The age attribute must be set to one of the following age groups: child, teen, adult, or senior. The gender attribute must be set to either male or female. The following example searches for an adult male voice, where the adult attribute is required and the male attribute is optional and speaks "Hello world":

```
set_age(adult,required),
set_gender(male,optional),
speak_with_attributes("Hello world").
```

3 Implementation

The development of LPA-Speech can be divided into two parts, the development of C++ functions to control SAPI, and creating the interface between LPA-Prolog and the C++ functions. LPA-Speech consists of a DLL (Dynamic Link Library) file containing a number of functions written in C++ which can be accessed externally by Prolog or any other programming language that provides support for DLL files and a companion Prolog file containing predicates to access the C++ functions.

3.1 Creating the DLL

Each Prolog predicate for either speech recognition or synthesis corresponds to a C++ function in the DLL file. The DLL files is written just like any other C++ program except that the functions which are to be accessed externally by Prolog are declared as follows:

```
extern "C" __declspec(dllexport) char * lpa_listen();
```

In this example `lpa_listen()`, which implements the predicate `listen`, takes no arguments, but returns a character list, which in this case will be returned to Prolog in the form of a memory address. Any other functions used within the DLL file but not used externally are declared and defined as usual.

3.1.1 Speech Recognition

Before speech recognition is able to occur, the recognition engine and grammar must be initialized. Once this is accomplished, the method for listening is quite simple as can be seen below from this piece of the `lpa_listen()` function:

```
CComPtr<ISpRecoResult> cpResult;

while (SUCCEEDED(hr = BlockForResult(cpRecoCtxt, &cpResult))) {
    cpGrammar->SetDictationState(SPRS_INACTIVE);
    CSpDynamicString dstrText;
    char * mystring;
```

```
    if (SUCCEEDED(cpResult->GetText(SP_GETWHOLEPHRASE,
            SP_GETWHOLEPHRASE, TRUE, &dstrText, NULL))) {
        mystring = dstrText.CopyToChar();
        cpResult.Release();
        return mystring;
    }

    cpGrammar->SetDictationState(SPRS_ACTIVE);
}
```

This code loops until the recognition engine has finished and returns a result to `dstrText`. The text in `dstrText` is then copied to the character list `mystring` which is returned to Prolog as an address in memory.

3.1.2 Speech Synthesis

The following function, `lpa_speak_no_attr`, takes a character list as input and speaks it using the default voice settings for SAPI:

```
void lpa_speak_no_attr(char * mystring) {
    ISpVoice * pVoice = NULL;

    if (FAILED(::CoInitialize(NULL)))
        return;

    HRESULT hr = CoCreateInstance(CLSID_SpVoice, NULL, CLSCTX_ALL,
        IID_ISpVoice, (void **)&pVoice);

    if(SUCCEEDED(hr)) {
        hr = pVoice->Speak((CSpDynamicString)mystring, 0, NULL);
        pVoice->Release();
        pVoice = NULL;
    }

    ::CoUninitialize();
}
```

This method begins by creating an instance of type `ISpVoice` called `pVoice`. Once `pVoice` is initialized, the method simply casts the character list into a

SAPI defined type called `CSpDynamicString` and feeds it to `pVoice` which speaks the input.

SAPI provides users with the ability to control certain characteristics of speech output. In this way, it is possible to override the default settings and select the specific type of voice to be used. Control over these characteristics falls into two categories, basic output control and attribute-based voice selection.

`lpa_speak` like `lpa_speak_no_attr` takes a character list as input and speaks it, but also takes several other arguments that correspond to the settings SAPI allows the user to modify. As mentioned previously these settings include rate of speech, volume, gender, and age. The following example taken from `lpa_speak` illustrates how to set the rate of speech to its highest speed.

```
hr = pVoice->SetRate(10);
```

Setting the volume works in an similar way.

The age and gender attributes are modified in a slightly different way. Since SAPI usually comes standard with a variety of different voices it also provides a way to control what voice gets chosen for speech output. This is done through the use of attributes. Attributes are information relating to each voice that is stored in the Windows Registry. These attributes can be used to search through all of the available voices and choose the one that best matches. This is done using a provided method called `SpFindBestToken`. When using `SpFindBestToken`, there are two categories of attributes that are used, required attributes and optional attributes. `SpFindBestToken` returns only voices that match all of the required attributes. If `SpFindBestToken` is unable to find any voices that match all of the required attributes, the default voice is chosen. Optional attributes on the other hand, only influence the voice that is chosen. Thus, if there are two voices that match all of the required attributes, the one with the most optional attributes that are also met is the one that is returned by `SpFindBestToken`. A call to `SpFindBestToken` appears in the following example:

```
hr = SpFindBestToken(SPCAT_VOICES, required, optional,
    &pCurVoiceToken);
```

When `SpFindBestToken` is called, the attributes in the required and optional variables must be in the following format:

```
"Age=Adult"
```

3.2 Prolog Interface

Once the DLL file has been created, it must be loaded into LPA-Prolog so that the C++ functions are available for use. This is done using the built-in predicate `winapi/4` as follows:

```
winapi((kernel32,'LoadLibraryA'),[`lpa_speech.dll`],0,Handle)
```

Now the C++ functions mentioned in the previous section are available for use within Prolog.

To call a C++ function in the DLL file in Prolog the `winapi/4` must be used. For example, a call to `lpa_listen()` is performed in Prolog as follows:

```
winapi((lpa_speech,lpa_listen),[],0,Address)
```

This tells Prolog to call `lpa_listen` from the `lpa_speech` DLL with an empty list of arguments and to return any output to `Address`. Once the memory address to the output character list is returned to Prolog the built-in predicate `wintxt/4` can be used to return the string as follows:

```
wintxt(Address,0,0,String)
```

Calls to `lpa_speak_no_attr` are performed similarly with a couple of exceptions. `lpa_speak_no_attr` requires an input in the form of a character list to be spoken and no value is returned. These changes can be noted in the following call to `lpa_speak_no_attr`:

```
winapi((lpa_speech,lpa_speak_no_attr),[Sentence],0,_)
```

Calls to `lpa_speak` require even more arguments to be supplied as show here:

```
winapi((lpa_speech,lpa_speak),[Sentence,Rate,Volume,Required,
    Optional],0,_)
```

As before `Sentence` is a character list. Both `Rate` and `Volume` are integers. As mentioned in the previous section `Required` and `Optional` are character lists in the format of `"Age=Adult"`. Again, no value is returned.

References

[1] FunctionX. *Win32 Static DLL*. Available online at http://www.functionx.com/visualc/libraries/staticdll.htm.

[2] Logic Programming Associates. *Documentation Files*. Available online at http://www.lpa.co.uk/dow_doc.htm.

[3] McClain, Jonathan (2003). *SWI-Speech: An Interface Between SWI-Prolog and Microsoft SAPI*. University of Georgia. Available online at http://www.ai.uga.edu/mc/ProNTo.

[4] Microsoft. *Microsoft Speech Technologies Website*. Available online at http://www.microsoft.com/speech.