# SWI-Speech: An Interface Between SWI-Prolog and Microsoft SAPI

Jonathan T. McClain
Artificial Intelligence Center
The University of Georgia
http://www.ai.uga.edu

May 8, 2003

**Abstract**

This paper describes the implementation of *SWI-Speech*, an interface between SWI-Prolog and Microsoft SAPI (Speech Application Programming Interface). SWI-Speech is a component of *PRONTO* (Prolog Natural Language Toolkit), a package created by the Artificial Intelligence Center at the University of Georgia. Topics include a discussion of how to create and control foreign libraries from SWI-Prolog, as well as a discussion of controlling SAPI using C++.

## 1 Introduction

This paper describes the implementation of *SWI-Speech*, an interface between SWI-Prolog and Microsoft SAPI (Speech Application Programming Interface). SWI-Speech is a component of *PRONTO* (Prolog Natural Language Toolkit), a package created by the Artificial Intelligence Center at the University of Georgia. It is fully compatible with SWI-Prolog version 5 and Microsoft SAPI version 5.

# 2 Motivations for interfacing SWI-Prolog and SAPI

Prolog is extremely useful to those involved in programming applications for processing natural language. Covington (1994), outlines a number of reasons why Prolog is considered the most suitable language for natural language processing (NLP), in particular:

1. Prolog provides a way to manipulate large and complex data structures, such as lists of words, easily and efficiently.

2. A program written in Prolog is capable of modifying itself, allowing the use of abstract programming methods.

3. A search algorithm, breadth-first search, is built into Prolog and is easily used in a number of natural language parsers.

4. Unification is also built into Prolog and can be used to build data structures in a way that the order of processing does not matter.

Because of Prolog's suitability for NLP, it would be strongly desirable to manipulate language synthesis and recognition programs directly from Prolog, where all the processing is done. Unfortunately, a number of these sorts of packages are designed to be used by more conventional programming languages such as C or C++, and it is difficult or impossible to access them from other languages. In many cases, it is necessary to write methods in another language to use these applications and then call them from a Prolog program. Thus, a standard interface to applications of this sort would greatly accelerate the development of NLP applications that require the recognition and synthesis of language.

Microsoft SAPI is a little known speech recognition and synthesis engine that is supported in all versions of Windows after Windows 95. The Microsoft Speech SDK (System Development Kit) version 5.1 is available for free download from the Microsoft Speech Technologies Website. Speech SDK 5.1 is compatible with a number of programming languages, but most of the documentation focuses on examples written in C++. In general, any programming environment that supports OLE automation will work for writing SAPI applications.

# 3 SWI-Speech implementation

The development of SWI-Speech can be divided into two parts, the development of C++ methods to control SAPI, and creating the interface between the methods and SWI-Prolog. C++ was chosen because it is ideally suited for both of these tasks. The general architecture of SWI-Speech consists of a DLL (Dynamic Link Library) file containing a number of Prolog-accessible predicates written in C++, and a companion Prolog file containing methods to control these low-level methods more easily.

## 3.1 Creating and controlling foreign libraries for SWI-Prolog

Creating a DLL file that is capable of being accessed by SWI-Prolog is facilitated by the C++ interface to SWI-Prolog provided by the file *SWI-cpp.h*. This file comes with SWI-Prolog and can be found in the *"include"* directory of an SWI-Prolog installation. SWI-cpp.h makes it easy to create predicates that are accessible by SWI-Prolog and are written in C++. The definition of `hello_world/1` a simple predicate defined using SWI-cpp.h, along with its output is included below (Wielemaker 2000):

```
PREDICATE(hello_world,1)
{
char * Name = A1;
cout << "Hello " << Name << endl;
return TRUE;
}

?- hello_world("Jon").
Hello Jon
```

This predicate takes a list of characters in Prolog and prints them to the screen. There are a few aspects of this predicate that are worthy of note. First, the arguments of `PREDICATE()` provide the name and arity of the predicate. The arguments of a defined `n`-place predicate are accessible using the macros `A1` through `An`. Finally, and most importantly, the statement `char * Name = A1` shows how a Prolog defined type is converted to a native C++ type. In this case, a list of characters is being converted to a character string. Much of the difficulty in writing predicates such as this lies in the

process of type-conversion. This is because while most standard types are easily converted, special types can be quite difficult since the conversions are not defined by SWI-cpp.h. A quick solution is to find a simple type that can be converted to both types and use it as an intermediary between them. For more information regarding the specifics of type conversion, see Wielemaker (2000).

After creating the DLL file containing predicates defined in C++, these predicates must be loaded into SWI-Prolog in order to use them. The provided predicate `load_foreign_library/1` makes this simple. However, when using `load_foreign_library/1`, one must be careful not to load the same library twice. The following predicate, `ensure_foreign_library_loaded/1` loads a library only in the case that it has not already been loaded:

```
ensure_foreign_library_loaded(ForeignLibrary) :-
  foreign_library_loaded(ForeignLibrary),
  !.

ensure_foreign_library_loaded(ForeignLibrary) :-
  load_foreign_library(ForeignLibrary),
  assert(foreign_library_loaded(ForeignLibrary)).
```

Thus, if the DLL is named SWI-Speech.dll, then the following query must be called before any of the predicates within it may be used.

```
?- ensure_foreign_library_loaded(swi-speech).
```

## 3.2 Controlling SAPI with C++

There were two primary methods that needed to be created in order to control SAPI via SWI-Prolog, a method to speak a line of text, and a method to control the speech recognition engine. Although the speech recognition engine was more difficult to get working at first, the speech synthesis method eventually became the more complicated of the two due to SAPI's capability of controlling voice output.

### 3.2.1 Controlling speech recognition in SAPI

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.

```
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 (A1 = mystring);
   }
   cpGrammar->SetDictationState( SPRS_ACTIVE );
}
```

This code continues to loop until the recognition engine has finished and returns a result to `dstrText`. The text in `dstrText` is then copied to `mystring` to be converted back to a list of characters in Prolog.

### 3.2.2 Controlling speech synthesis in SAPI

The following predicate, `swi_speak/1`, takes a character list in Prolog and speaks it using the default voice settings for SAPI.

```
PREDICATE(swi_speak,1)
{
  ISpVoice * pVoice = NULL;
  CSpDynamicString mystring;
  HRESULT hr;
  if (FAILED(::CoInitialize(NULL)))
      return FALSE;
  hr = CoCreateInstance(CLSID_SpVoice,NULL,CLSCTX_ALL,
                        IID_ISpVoice,(void **)&pVoice);
  if( SUCCEEDED( hr ) )
  {
      mystring = A1;
      hr = pVoice->Speak(mystring, 0, NULL);
      pVoice->Release();
      pVoice = NULL;
```

```
  }
  ::CoUninitialize();
  return TRUE;
}
```

This method begins by creating an instance of type `ISpVoice` called `pVoice`. Once `pVoice` is initialized, the method simply copies the Prolog character list into a SAPI defined type called `CSpDynamicString` and feeds it to `pVoice`. Luckily in this case, a Prolog character list easily converts to type `CSpDynamicString` and no intermediary is needed.

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

SWI-Speech provides Prolog programmers with control over two basic output characteristics, rate and volume. Both of these aspects are controlled directly via methods within the `ISpVoice` type, `SetRate` and `SetVolume`. The rate of speech must be set to a number between -10 and 10, with -10 being the slowest and 10 being the fastest. Volume must be set between 0 and 100 with 0 being off and 100 being the loudest. The following example illustrates how to set the rate of speech to its highest speed. Setting the volume works in an exactly similar way.

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

Since SAPI usually comes standard with a variety of different voices, and users also have the opportunity to obtain more voices from third-party providers, SWI-Speech 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);
```

SWI-Speech gives programmers control of two different attributes, age and gender. The age attribute can have four values, child, teen, adult, and senior. The gender attribute can be set to either male or female. When `SpFindBestToken` is called, the attributes must be in the same format as the following, `"Age=Adult"`. Thus, it was found that it was easiest to control the creation of these attribute lists within Prolog and then pass them to the DLL for processing.

# 4 Future Work

There are a few capabilities of SAPI that SWI-Speech does not currently address. First, SAPI gives programmers the ability to define new grammars for the speech recognition engine. At present, SWI-Speech only allows the use of the default grammar provided with SAPI. Although this grammar is quite extensive (the author has not yet found a word that it could not identify), this may not be acceptable in some special cases. SWI-Speech also currently does not address one searchable attribute of voices for speech synthesis, the vendor attribute. This attribute would allow programmers to search for voices that were created by a specific vendor.

All in all, SWI-Speech is a good tool that gives Prolog programmers access to the SAPI speech engine without having to deal with the problems of interfacing. This package would help speed up the work of anyone involved in NLP application development with SWI-Prolog.

# References

Covington, Michael A. (1994). *Natural Language Processing for Prolog Programmers*. Upper Saddle River, NJ: Prentice-Hall.

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

Wielemaker, Jan (2000). *A C++ Interface to SWI-Prolog* Available online at http://www.swi-prolog.org/packages/pl2cpp.html.