Overview

SRLanguageModeler helps one create, test and save SRLanguageModel objects. It can parse language model descriptions that have been written out in a special Backus-Naur Form (BNF) and produce SRLanguageModel objects. These can then be tested using the Speech Recognition Manager in a live recognition session to determine how well different utterances in the language model can be recognized and distinguished from one another. Finally, the SRLanguageModels can be saved to disk in either the data or resource forks of a file so they can be later loaded for use in an application.

SRLanguageModeler can save the developer hours of code development for those language models (and parts of language models) which do not change dynamically while the program runs. One of the strengths of the Speech Recognition Manager is that it can be used to build language models on the fly, for example to allow any of the visible hyper links in a World Wide Web browser window to be spoken. Still, many programs use a number of language models that are static. These can be most easily built, tested and saved using the SRLanguageModeler application.

BNF Description

The Backus-Naur Form language model description allows the specification of complex language models in a fairly compact way. One simple example is shown here:

```
<Call Someone> = Call <Person> at <Place>;
<Person> = Matt | Arlo | Brent;
<Place> = Work | Home | the office;
```

This specifies all the utterances

```
"Call Matt at Work"
"Call Matt at Home"
"Call Matt at the office"
.
.
.
"Call Brent at the office"
```

in 3 lines instead of 9.

In this example "<Call Someone>" is referred to as the *top level* or *root* language model, while the language models "<Person>" and "<Place>" are called *embedded* language models.

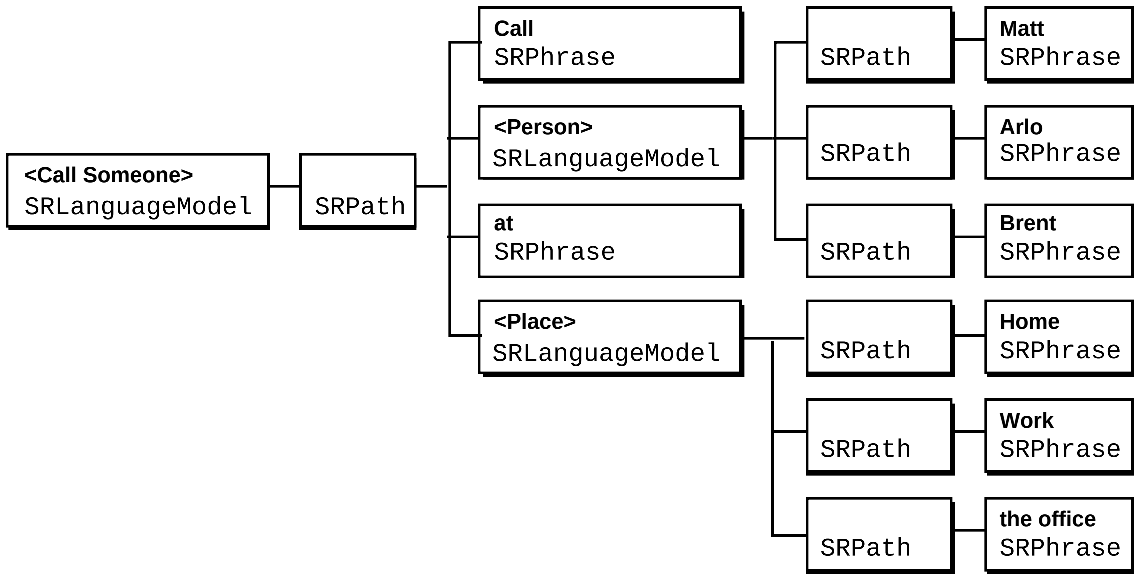
Parser Structure Rules

In order for a language model description to be successfully parsed by the SRLanguageModeler application, it must follow a few rules of structure and syntax. If the rules of structure or syntax are not followed, SRLanguageModeler will do its best to let you know where it had problems. The rules of structure are outlined here.

- There must be only one top level language model.
- Every language model must have one and only one definition.

Resulting SRLanguageModel structure

When SRLanguageModeler parses the simple language model description above, it produces an SRLanguageModel object with the following structure:



The type of each SRSpeechObject in the hierarchy is shown in the bottom of its box, while its spelling is shown on top. This SRLanguageModel hierarchy can be traversed using the Speech Recognition Manager routines SRCountItems and SRGetIndexedItem.

While this structure is not as dense as could be created by hand coding the Speech Recognition Manager calls needed to build this language model, it does reflect the structured way in which SRLanguageModeler builds a generalized language model. Namely,

- Each SRLanguageModel is made up of one or more SRPaths. These correspond to the parts of the right hand side of a language model assignment statement delimited by the '|' character.
- Each SRPath is made up of one or more SRPhrases and SRLanguageModels.

This regularity in structure makes it fairly simple to write code that traverses a language model produced by SRLanguageModeler. Your application can efficiently interpret recognition results by traversing a language model returned in the kSRLanguageModelFormat property of SRRecognitionResult objects sent to your application after each utterance.

Setting Properties

Traversing a language model is made even easier if the individual objects in the hierarchy have unique reference constants (hereafter called refCons) that you have assigned. Typically this would require a call to SRSetProperty with the kSRRefCon selector, but SRLanguageModeler lets you specify refCons right in the language model description. Setting the refCons in our simple example is done like this:

```
<Call Someone>    { kSRRefCon = 100 } = Call <Person> at <Place>;
<Person> =  Matt  { kSRRefCon = 200 } |
            Arlo  { kSRRefCon = 300 } |
            Brent { kSRRefCon = 400 };
<Place> =   Work  { kSRRefCon = 500 } |
            Home  { kSRRefCon = 600 } |
            the office { kSRRefCon = 700 };
```

Using this listing, the language model "<Call Someone>" is assigned a kSRRefCon property of 100. The SRPath containing the SRPhrase spelled "Matt" has a kSRRefCon property of 200. You'll see how to set the property on a SRPhrase below.

The kSRRefCon property is probably the property you'll use most, so we've provided a short hand for setting this property. Namely, we can rewrite the language model description above as

```
      <Call Someone> {100} = Call <Person> at <Place>;
      <Person> =  Matt {200} | Arlo {300} | Brent {400};
      <Place> =   Work {500} | Home {600} | the office {700};
```

with identical results.

There are a number of other properties that you can set on a SRLanguageObject (that's any part of a SRLanguageModel). Detailed descriptions of these can be found in the chapter "Speech Recognition Manager", but we'll cover a few of them right here.

Let's say we wanted to make the phrase "right now" in "Call <Person> right now" optional. That would allow the user to say, e.g. "Call Matt right now" or simply "Call Matt." Typically this would require us to set the kSROptional property of the SRPhrase "right now" to true. But SRLanguageModeler lets us specify this property right in the description as follows:

```
      Call <Person> (right now) { kSROptional=true }
```

Here we've indicated that we want to set the property on the phrase by enclosing it in parentheses. As with the kSRRefCon property, there is a short hand for the optional property; you can write the above expression as

```
      Call <Person> [right now]
```

with identical results. There is also a short hand expression for setting the kSRRepeatable property as well:

```
      Call <Person> [ad nauseum]+
```

This would allow utterances like "Call Arlo ad nauseum ad nauseum ad nauseum." Finally, you can combine the properties kSROptional and kSRRepeatable as follows. The long-hand form (for any list of properties) is:

```
      ... (ad nauseum) { kSROptional = true; kSRRepeatable = true }
```

while the short hand version is

```
      ... [ad nauseum]*
```

this allows 0 or more instances of "ad nauseum" to follow the utterance "Call <Person>."

Here are a few syntax rules about setting properties that you should remember:

- To set a property on a language model, a property list must follow the language model name on the left hand side of its definition:
  "<Call Someone> { kSROptional = true } = ..."
- To set a property on a path, the property list must appear at the end of the path. For example:
  "... Matt {2} | Arlo {3} Reeves ;" -->> Error
  "... Matt {2} | Arlo Reeves {3};" OK
  "... call <Name> now {10} | hangup {12};" OK
- To set a property on a phrase the phrase must be delimited by parentheses and followed by a property list. And remember that phrases can not contain embedded language models (unlike paths, like the "call <Name> now" path shown above):
  "(a phrase) { kSRRejectable = true; kSRRejectionLevel = 50}" OK
  "(a <lm> phrase) { kSROptional = true }" -->> Error
- Currently, if you want to set several properties on a phrase, you must use an explicit property list; no short hand for the kSROptional and kSRRepeatable properties is allowed. For example:
  "an ([optional]) {100} word" -->> Error
  "an (optional) { kSROptional=true; kSRRefCon=100 } word" OK

Creating a ".h" interface

To make it possible to keep all of your refCon definitions (#define's) in one place, we've made it possible for you to re-use your language model description file as a C style interface (".h") file. Basically, you can have a simple ".h" file which starts with a list of #define's describing your reference constants. Then the body of the language model description follows, delimited by C style comments ("/*" and "*/"). You can then use the refCons you've #define'd in both your language model description and in the code you use to traverse recognition results. This capability places several restrictions on the syntax you use in the .h file:

- Your #define statements cannot contain any expressions that need evaluation.
- Enums are not currently supported.
- C++ style comments ("//") are OK, but anything between C style comments (except the C++ comments) will be interpreted as a language model description.

Examples

We've included a couple of example language model files that you can parse with the SRLanguageModeler application. They're simply text files the contents of which follow the structure and syntax rules outlined in this document. To use them follow these steps:

o Launch the SRLanguageModeler application

o Choose "New Mic Recognizer" from the File menu.

o Choose "Open LanguageModel Text…" from the 'LanguageModel' menu.

o Select the example file "LMSample.h". This will cause the text file to be loaded and parsed into an SRLanguageModel object that will appear at the bottom of the LanguageModel menu.

- o Choose "Paste Current LanguageModel" from the LanguageModel menu to see the language model description and get an idea of what you can say.
- o Click the "Start Listening" button and try out some utterances!