Building speech interaction systems with Perl and free software

David Huggins Daines, Cepstral LLC, <dhd@cepstral.com>
Kevin A. Lenzo, Alan W Black, Cepstral LLC, Carnegie Mellon University

June 9, 2001

```
http://www.cepstral.com/~dhd/yapc19101/
http://www.cepstral.com/source/


* What kinds of things can I do with speech?
  - Telephony applications
  - Remote access
  - E-mail, web
    * Over the phone or on PDAs
  - Desktop control
    * Opening applications, window management functions
  - Accessibility
    * Screen readers
    * Assistive devices
  - More sophisticated desktop/application control
    * Dictation (maybe)
    * Robots, robots, robots

* What speech software is out there?
  - Non-free stuff
    * Synthesis products are mostly geared towards telephony and
      server markets
    * Mac OS comes with good built-in speech synthesis that is
      accessible from MacPerl
    * For recognition, IBM ViaVoice is available gratis for individual
      use on Linux, and people have built nice stuff around it
  - And now, the good news: free stuff
    * Festival and Festvox
      - Festival is a platform for research and development of
        text-to-speech and synthesis, which also comprises a run-time
        synthesizer
      - It is written in C++ and has an embedded Scheme interpreter
```

      - Festvox consists of (rather extensive) documentation and a
        suite of tools for building voices that run under Festival
      - Currently, there are free English (UK and US) diphone voices
        that are free, as well as voices for several other languages
        that are gratis.
      - Festival can also use the MBROLA synthesizer for waveform
        synthesis, which has much higher quality and a wider range of
        languages. It's also "free as in beer only".
      - For specific applications, very good results can be achieved
        with "limited domain" voices, which are quite easy to build
        using the Festvox tools.
    * Sphinx
      - The Sphinx project has produced several generations of speech
        recognition systems, which are being released as free
        software.
      - Sphinx-II is the only one which is currently practical for
        production use.
      - Like Festival, Sphinx started life as a research system
      - Documentation is still pretty sparse, and the public release
        should be considered alpha-quality
      - Unlike commercial recognition software, there are no included
        language models
* More on this below

* Perl and speech
  - Speech applications must handle many different types of data.
    * Text
      - Most speech applications use text as the starting or ending
        point (i.e. text-to-speech, speech-to-text)
    * Audio
      - Binary data, which is easily accomodated by Perl scalars
    * Linguistic information
      - Structured data, which often consists of key-value pairs and
        nested data structures, and can be represented by hashes or
        opaque objects
      - Utterance structures (result of text analysis)
      - Phonetic information (segment, phrase structure)
      - Semantic frames for concept-to-speech
  - High-level languages make this much easier to deal with
  - Perl is also relatively fast, and very fast at doing hacky
    string frobbing
    * Speech involves a lot of hacky string frobbing

* Speech and audio.
  - Obviously, speech input and output require audio input and output
  - This doesn't have to be on the same machine as the processing, however

- "On-line" applications have different requirements than "off-line" ones
- Examples of off-line applications:
  * Playing MP3s
  * Command-line audio recording and playback tools
  * Desktop event sounds, gaming
  * Reading e-mail
- Examples of on-line applications:
  * Playing movies
  * Playing MP3s with synchronous "eye-candy"
  * Audio recording and playback with visual feedback or analysis
- On-line apps need control and information about the state of playback
  * Audio devices all buffer data, to different extents
  * Therefore all audio I/O is at some level asynchronous
  * Either you can block while waiting for things to happen and use threads to do other stuff at the same time, or you can make your entire program event-driven
    - The former does not work very well in Perl, so therefore we use POE

* Perl, speech, and audio
- Depending on your application, you may only care about speech
- Speech software will happily take over the audio device for you
- This may or may not be what you want it to do
- The same on-line/off-line distinction applies to speech applications
  * Barge-in and visual feedback require on-line processing
  * Recognition is also on-line by definition

* Talking
- The polluted Speech:: and Festival:: namespaces
  * Currently there are three modules for using Festival, all slightly different
  * Festival::Client is very basic and oriented towards doing text-to-speech only
  * Speech::Festival and Festival::Client::Async allow you to evaluate arbitrary Scheme code and feed back results and waveforms, which is necessary for doing barge-in and such.
  * Festival::Client::Async is my fault; it is designed for easy event-driven "asynchronous" operation, and thus integrates well with POE and other event loops.
  * All of them talk to a Festival server, so you have to have one running
    - festival --server &
    - I've got an init script that runs it at boot (as 'nobody')
      * In this case though it won't be able to access audio directly
- Festival as a library
  * This is possible, and I'm planning to do it soon, as it is needed for the total Perl-ification of Festvox

    * Festival is built on top of Speech-Tools, for which I have
      pre-alpha-quality Perl bindings which I'll release any day now...
  - Flite
    * Flite is a small, fast run-time-only synthesizer currently in
      development at CMU
    * It is designed from the ground up to be a reusable library,
      written in C, so bindings for Perl and other languages will be
      much cleaner than the Speech Tools and Festival ones
    * It will be a separate "target" for the Festvox build tools
    * Expected to be released any day now (...)

* Listening
  - Speech::Recognizer::SPX
    * Wraps the Sphinx-II utterance processing and search functions
    * "Procedural" interface
    * Accepts raw audio data, generates text search results
  - Utterance segmentation
    * To obtain the best results, it's necessary to delimit utterances
      - Done simply by calling uttproc_begin_utt() and uttproc_end_utt()
      - There are various strategies for figuring out when to do so
      - For close-talking microphones or telephones, you can use
        silence filtering since there isn't going to be a lot of
        ambient noise.
      - Ideally, for all situations, you would listen continuously and
        use the recognizer to distinguish speech from non-speech
        sounds
        * We don't have this working yet :-)
      - Push-to-talk works best for "desktop" stuff
      - Apple uses a "trigger" word that causes the recognizer to
        attend to commands
    * Audio::SPX::Continuous
      - Simple automatic threshold-based silence filtering
      - Meant to work with the rather clumsy Sphinx audio input module
      - Special hacks to allow it to be used with raw data from Perl
* Audio::SPX::Continuous->init_raw, init_nbfh
  - Language models and modelling
    * Good continuous speech recognition is absolutely dependent on
      higher-level information, i.e. knowing the sort of things that
      people are likely to say
    * Usually this is done by statistical language modelling,
      typically based on a representative collection (corpus) of
      possible sentences that would be recognized
    * VoiceXML "grammars" are the same thing
    * In many applications, you will have several different models and
      switch between them based on the state of the system
    * Language modelling is currently the Achilles' heel of the Sphinx

system, as the tools for building language models are not readily available or reliable

* Talking and listening
  - POE components for synthesis and recognition
    * Still somewhat of a work in progress
    * Work on post/subscribe basis using separate sessions sending messages
    * Recognizer, Silence filter, Synthesizer, Audio In/Out
  - In the simplest case, one can just create all the necessary components, tell them to "attach" to each other's events, and watch them go.
  - Audio control is done using notification events
    * Notifications can be set at arbitrary points in the data stream
    * They are forwarded to all sessions "attached" to the audio component
    * Listeners can force the audio to stop immediately
  - The principal problem is doing barge-in
    * Barge-in means reacting to new speech input while talking
    * In order for it to seem more "natural", you must have the ability to stop speech on word or phrase boundaries
    * This (currently) requires the application to handle audio output rather than the synthesizer, because the synthesizer only operates in "batch" mode
      - A more sophisticated Festival/Flite server may fix this
      - Or, using them as a library

* See Also
  - Festival project at http://cstr.ed.ac.uk/projects/festival/
  - Festvox project at http://festvox.org/
  - Sphinx project at http://www.speech.cs.cmu.edu/sphinx/ and http://sourceforge.net/projects/cmusphinx/
  - Cepstral's developer site at http://www.cepstral.com/website/pages/cep_developers.html