V-REX – VOICE RELEVANCE ENGINE FOR XFINITY

Stefan Deichmann, Oliver Jojic, Akash Nagle, Scot Zola, Tom Des Jardins, Robert Rubinoff, Amit Bagga
Comcast Labs

Abstract

V-REX is a new platform Comcast is building to provide speech-based applications for television control and other areas. V-REX applies automated speech recognition, natural language processing, and action resolution modules to interpret the user's request and identify the appropriate response. We describe here how we use V-REX to support an iPhone/Android app that allows users to control their cable set-top boxes by speaking into their phones. The primary focus of the work involves building grammar rules and dictionary entries for the range of requests the app can handle. We use the grammar and dictionary both to guide ASR and to allow NLP to extract the actions and entities in the request. We then convert these results into appropriate database queries that extract the information the user needs.

INTRODUCTION

Using a voice interface provides two advantages over traditional set-top-box remote or web interfaces. First, it eliminates the need for typing or other keyboard-based or remote-based text entry methods. (In the case of TV or cable remotes, this can be extremely tedious.) Furthermore, by allowing the use to directly specify what they want, it eliminates the need to wade through a series of menus or pages to find the desired option.

In order to provide a voice interface, we need to answer three questions: what words did the user speak, what action or information are they requesting, and how do we carry out the action or get the information? Each of these questions is answered by a specific module in V-REX. Automated speech recognition (ASR) identifies the words the user has spoken. Natural language processing (NLP) figures out what action or information has been requested. Action resolution (AR) responds to the request.

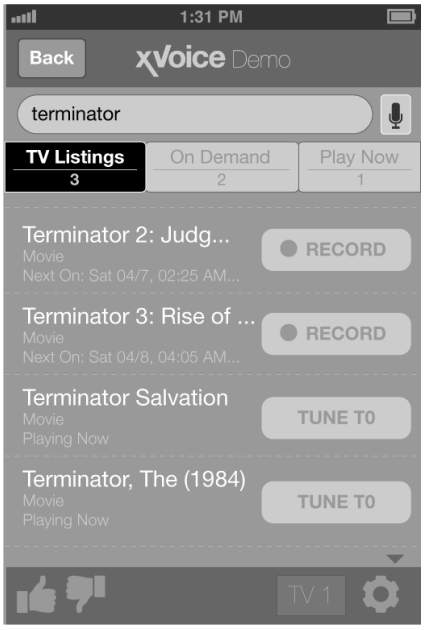

Figure 1 – Sample Results Display

Our initial V-REX application is an iPhone/Android app for Comcast customers that allows them to look up programs and control their set-top box. The app currently can handle three kinds of requests:

1) "What's on" – list what programs are available on a particular channel and/or at a particular time (including right now). The user can also specifically ask for sports games, or for a particular sport such as baseball or basketball.
2) "Tune" – switch the cable box to a specific channel
3) "Search/Find" – find when and on what channel a particular program is playing; this will also show if it is available in the On Demand library

For "what's on" and "search" requests, the results are displayed on the screen, and the user can select individual programs or channels to get more detail. For example, Figure 1 shows the response when the user asks "What's on CNN tonight"? In subsequent sections, we will describe each step of the process that produces this response.

AUTOMATED SPEECH RECOGNITION

The ASR module is built with CMU's Java-based toolkit, Sphinx4. We used Sphinx because it is a well-developed open-source system that we already had experience with.[1] We replaced Sphinx4's default acoustic model with one from VoxForge, a web site that collects transcribed speech for use with open source speech recognition engines. We built our own application-specific language model, which has two major parts, a pronouncing dictionary and a grammar, incorporating as well a general language model built on the English Gigaword Corpus (www.keithv.com/software/giga).

The dictionary maps words to their possible pronunciations at a phonemic level. The phonemes in our pronouncing dictionary are based on the ARPABet developed for speech understanding systems in the '70s. There are 39 phonemes, not counting variants due to lexical stress. Anything a person can say, including actions like "tune to" and channel names like ESPN, must be stored as a phonetic representation.

The grammar is a textual description of the combinations of words and phrases the system will accept. It is written in Java Speech Grammar Format, an augmented BNF-style format [1]. The grammar contains rules describing how users can ask to change a channel, what exactly counts as a title, and how people can ask about a time of day. Within a limited domain like television, the language model provides a higher level of accuracy in detection than it would normally achieve in an unconstrained system.

The dictionary and grammar are designed around three specific requirements of the application. The first is to handle channel names, many of which are not in the general vocabulary of the basic Sphinx system. For example, "SyFy" and "Tru TV" need to be added. In addition, some channel names may have multiple pronunciations, e.g. "Univision" can be spoken with either English or Spanish pronunciation; and some channel names may contain words that are in the general vocabulary but not commonly used together outside of the domain, e.g. "Fox Business" or "Showtime Family". To handle these cases, we added the names of all of the channels Comcast provides to the dictionary. Including the channel names in the dictionary does more than just improve recognition of these terms; it also lets us use a more precisely tailored language model, improving the overall ASR performance.

The second requirement is to handle a wide range of time and date specifications. While most of these are part of the general language, there are some that are specific to the television domain (e.g. "prime time"). More importantly, we want to directly handle complex phrases such as "next Tuesday evening after 10" and recognize them as indicating the time of a program as early in the processing as possible. To that end, we include in the grammar a large range of ways of referring to dates and times. A portion of this grammar is shown in Figure 2.

Finally, we need to recognize titles of movies and TV programs. This is a particular challenge, as titles can contain deliberate misspellings or ungrammatical phrases that would be rejected by a general language

[1] We are continuing to evaluate other ASR systems, but so far have found Sphinx4's performance and accuracy to meet our needs.

```
<temporalAdverb> = (   <weekday>
                     | on <weekday>
                     | this <weekday>
                     | prime time
                     | [right] now
                     | tomorrow <daytime>
                     | today <dayTime>
                   );

<dayTime> = ( morning | afternoon | evening | night );
<weekday> = (sunday | monday | tuesday | wednesday | thursday | friday |
saturday);
<clockTime> =
      (  at  <hour>  o'clock
      |  at  <hour> [ <minute>  ] [ <amOpm>  ]
      );
```

Figure 2 - A portion of the date/time grammar

model, e.g. "eXistenZ" or "De-Lovely", or subtitles that don't fit into normal sentence structure, e.g. "Dodgeball: A True Underdog Story". Even without these kinds of problems within a title, there is the danger of processing the title as a normal part of the whole sentence. For example "What time is Seven on?" is most likely asking about the movie "Seven", not channel 7 or seven o'clock. In order to handle these problems, we have added movie and TV show titles to our dictionary. This allows us to recognize titles when spoken (in places where titles make sense).

In order to add titles to the dictionary, though, we need to know which titles to add. We can't simply add **all** of the titles that have ever been produced, because that would involve several million titles. This would drastically increase the size of the dictionary, seriously diminishing both speed and accuracy of the ASR system. Furthermore, the vast majority of the titles would be for movies or TV shows that aren't currently available and that the user has almost certainly never heard of. Instead we limit the titles to all shows that are currently available (either on a broadcast or cable channel or on demand). We also include the most popular movies and TV shows, even if they are not available, since there is a good chance the user will ask for them.

The top level of the grammar specifies the range of possible requests the system can handle (and therefore needs to recognize). A simplified version of the grammar is shown in Figure 3. The three possible request types each have their own top-level rule, which is further specified in subsequent rules. The various parts of the grammar are combined to provide a language model that constrains and guides the recognition process.[2]

[2] As the application expands to handle a larger range of requests, we anticipate allowing some of the ASR work to use a more general statistical language model, so that the system can recognize unrestricted language. This will be particularly important when we start handling extended dialog. The grammar will stay play an important role in interpreting the request, though, as described in the section on the NLP module.

```
<whatsOn> = <actionPhrase> [ <modifierPhrase> ];
<tuneTo> = <tuneToPrefix> <channel>;
<search> = (<searchPrefix> <title> [now] | <title>);

<searchPrefix> = ( can i watch | play | search | find );

<actionPhrase> = <whatsOnPrefix> | <whatsOnPrefix> <channel>;

<tuneToPrefix> = ( (tune to | change the channel [to] | change [to] );
```

Figure 3 – a portion of the top level grammar

NATURAL LANGUAGE PROCESSING

Once the ASR module processes the user's request, the output (i.e. the request in text form) is sent on to the NLP module. NLP starts by parsing the text, using the same grammar used by ASR.[3] The text is parsed using the JSGF parsing facility, part of the package used to write the grammar (as described above). The NLP module uses the resulting parse tree to interpret the utterance, inferring the semantics from the rules used and the tags assigned in the parsing process.

For example, consider the request "What's on Disney on Saturday?"; the parse structure for this is shown in Figure 4. Here we can determine the request type from the <whats-on-phrase> node, the requested channel from the <channel> node, and the time constraint from the <temporal-adverb> node. These three different pieces of information are actually obtained in three different ways. The request type is determined to be "what's on"

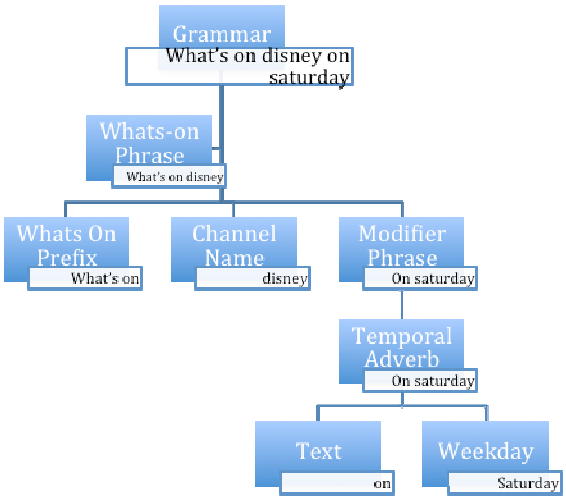

Figure 4 - Parse Structure for "What's on Disney on Saturday?"

simply from the presence of a <whats-on-phrase> node, which reflects that the request has the structure and content of a "what's on" request. The channel is determined to be "Disney" based on the value of the <channel-name> node, which the grammar indicates is the appropriate value for the text "Disney". (In this case the channel name and the text are the same, but that is not the case for all channels.) The time constraint needs more complex processing, which is provided by special-purpose code in the NLP module that knows the range of possible parse structures and how to extract time and date values from them. Once it has identified all the pieces of the request, the NLP module assembles them into a request structure that is passed on to the AR module.

[3] The two modules don't have to use the same grammar, although that is the case in the current system. It might be appropriate to use different grammars if, for example, we want to allow incidental comments that are not relevant to the request (e.g. "tune to HBO, please"). In particular, if we switch to using a statistical language model for part or all of ASR rather than a grammar-based one, NLP and ASR will need to use different grammars.

For the current application, we assume that any information not indicated in the request is deliberately left unspecified. For example, if no channel is mentioned, we assume the user wants to know the most popular shows available on any channel in the requested time span; if no time is specified, we assume the request is for programs on during prime time today. Missing information is therefore either left unspecified in the request or filled in with a default value.

In more complex applications, we might need to explicitly mark that the information is missing so that later processing can take necessary action to deal with the situation.

ACTION RECOGNITION

The AR module receives the request structure built by the NLP module and attempts to carry out the request. This involves constructing and sending an appropriate query to Comcast's REX search system. REX is the system we use to index and search through the complete set of programs available on broadcast and cable channels and on demand. The precise form of the query to REX depends on the type of action requested. For "what's on" requests, the query indicates the requested channel (if any) and time span. For "search" requests, the query indicates the title that the user specified. For "tune" requests, the query indicates the name of the requested channel. We need to query REX for tune requests to find the channel number corresponding to the channel name; if the request specified a channel number directly then we can skip the query.

The results returned by REX are packaged up along with an indication of the request type and the output of the ASR module and sent back to the client app. In case of an error, an appropriate error code is returned, along with any relevant information about the error. As mentioned above, a more complex application might require further interaction with the user, either to resolve an error or to get more information needed to carry out the request. The AR module contains a simple text-to-speech component, based on the FreeTTS system [2], to handle such interaction. This capability is not needed currently, though.

THE IPHONE/ANDROID CLIENT APP

The server-side components described above support a client iPhone/Android app that allows the user to speak requests and see and respond to the results. The initial screen for this app is shown in Figure 5; the user can press the microphone and speak a request. A typical response is shown in Figure 6; here the user has asked "what's on CNN tonight?" and the app displays the list of programs returned after processing through the ASR, NLP, and AR modules. The user can select a specific show to get more detail, as shown in Figure 7. Search requests display similar responses, except that the results are organized by relevance to the request rather than by time.

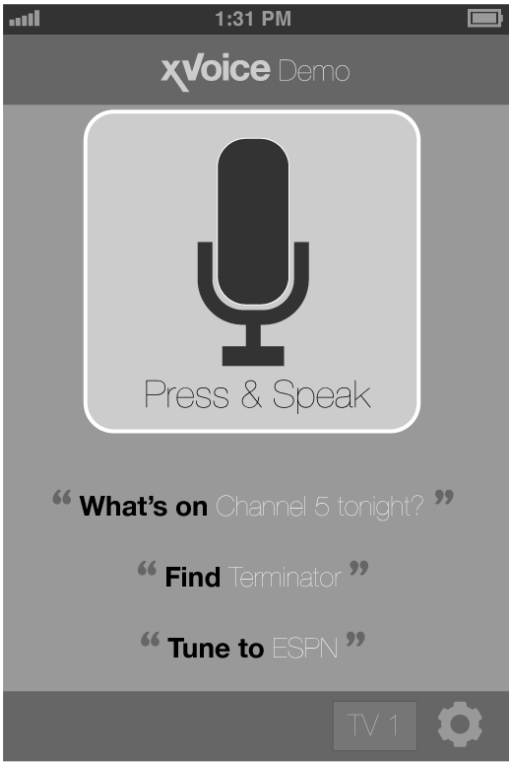

Figure 5 – Initial Client App Screen

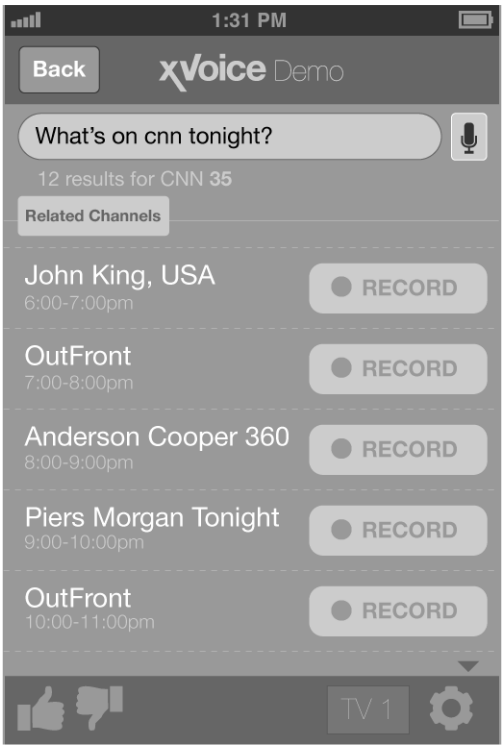

Figure 6 – Response to "What's on" request

Figure 7 – Details of a specific program

CONCLUSION AND FURTHER WORK

Speech-based interfaces provide a uniquely simple and direct interface. Users can simply say what they want, without having to type in complex queries or navigate through layers of menus. The V-REX platform combines automated speech recognition, natural language processing, and action resolution to power speech interfaces. Our initial app brings this ability to searching and controlling cable set-top boxes. We are exploring ways to extend V-REX to other applications built on Comcast's cable/internet infrastructure.

One possibility is to extend it to our Play Now system for watching programs over the web. A more interesting extension is to apply it to Comcast's home security service, so that people can easily check on the status of their homes while they are away.

REFERENCES

[1] http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html

[2] http://freetts.sourceforge.net/docs/index.php