VoiceXML Adventure Game

Jonathan Eisenzopf

March 8, 2001

Continuing from Part I [http://webref.com/perl/tutorial/20] of our comprehensive series of VoiceXML tutorials, in Part II we will develop a simple, but fully dynamic VoiceXML application in Perl for the Tellme.com voice portal.

Overview

```
"You are in an open field west of a big white house with a
boarded front door. There is a small mailbox here."
```

Do you remember the original Zork text adventure game and Choose Your Own Adventure books? Maybe you're even old enough to remember the original *Adventure* game. If you don't, you're missing out. We're going to develop a simple Zork-like application using VoiceXML. In Part III, we will create a more advanced version that utilizes its own XML format for building text adventures.

Examples

As we step through each code example, you will also be able to dial into Tellme and demo the code live on the Mother of Perl Tellme extension. Simply dial 1-800-tellme, wait for the Tellme main menu, then dial 1-73759. You will hear a greeting and a menu of options. Dial *2* for Tutorial 21 and then press the number that corresponds with the example in the tutorial. For example, when you see:

Example 1.

you can interact with the source code by pressing 1 on your phone. Each example is also linked to the XML source code file, where you can examine the file in your Web browser, or launch it directly into your favorite XML editor.

In addition to the live examples, you will find the source code for all examples in the tutorial21.zip [tutorial21.zip] file, which you can use for free to create your own VoiceXML applications.

Creating a Grammar for DTMF Input

Grammars

Before we proceed any further, we need to talk a bit more about grammars. They are a very important part of any VoiceXML document. A grammar is a set of expressions that define what DTMF tones and spoken input will be recognized by the system. These grammars are limited to what you program them to recognize. Depending on where you place the grammar, the scope can be the whole document, a form, or a field within a form. The Tellme grammar syntax uses the Nuance Grammar Specification Language (GSL).

It's not too difficult to define a simple grammar that allows a user to make a selection via DTMF tones. Let's define a simple grammar that provides our users with the option to open one of three doors in our adventure.

```
<grammar><![CDATA[
        [
          [dtmf-1] {<option "door1">}
          [dtmf-2] {<option "door2">}
          [dtmf-3] {<option "door3">}
        ]
      ]]>
</grammar>
```

First, grammars are placed within the *grammar* element. You should also surround the grammar with a *CDATA* section, which tells the XML parser not to process the text inside. In the grammar above, we have defined three options. Each option is defined by a grammar statement. The statement consists of an expression to be matched, and the corresponding value to return when a match occurs.

The expression used to match a digit is: `dtmf-$` where `$` is the number we want to match. For example, `dtmf-1` would be matched if a user presses 1 on their telephone keypad.

The second part of the expression defines the value that will be returned when we get a match. For example, `{<option "foo">}` will return the text `foo` when the user input matches the first part of the expression.

Next we're going to apply this simple grammar to a real VoiceXML document.

Getting DTMF Input

The first version of our adventure game below will only accept DTMF input. The user can select 1, 2, or 3, which corresponds with one of the three doors.

An example user session follows:

Tellme: You are in a small room with three doors.

Tellme: To open the first door, press 1.

Tellme: To open the second door, press 2.

Tellme: To open the third door, press 3.

User: (pressed 1)

Tellme: You see a large hungry monkey.

Now, let's take a look at the VoiceXML document.

Example 1. [example1.xml]

```
<?xml version="1.0"?>
 <vxml version="1.0">
   <form>
     <field name="answer">
       <grammar><![CDATA[
           [
             [dtmf-1] {<option "door1">}
             [dtmf-2] {<option "door2">}
             [dtmf-3] {<option "door3">}
           ]
         ]]>
       </grammar>
       <prompt><audio>You are in a small room with three
 doors.</audio>
       <pause>300</pause> <audio>To open the first door, press
 1.</audio>
       <pause>300</pause> <audio>To open the second door, press
 2.</audio>
       <pause>300</pause> <audio>To open the third door, press
 3.</audio> </prompt>
     </field>
     <filled>
       <result name="door1">
         <audio>You see a large hungry monkey.</audio>
                <reprompt/>
       </result>
       <result name="door2">
         <audio>You see another room with three doors, a man, and his
 monkey.</audio>
         <reprompt/>
       </result>
       <result name="door3">
```

```
        <audio>You see a man scratching his
 monkey.</audio>
        <reprompt/>
      </result>
    </filled>
  </form>
 </vxml>
```

For more information on using the *filled* and *result* elements, please see Page 3 [http://webref.com/perl/tutorial/20/3.html] of the first part of this tutorial. You might have also noticed the *pause* element in the example above. Using the element forces Tellme.com to pause between prompts. In this case, we want to stop for 300 milliseconds between options to duplicate the pause that would occur in natural speech.

Adding Voice Selections

Now that we have a simple interactive application working, we need to add the ability for users to provide voice commands in addition to DTMF input. This is done by adding additional grammar expressions.

```
  <grammar><![CDATA[
          [
            [dtmf-1 one] {<option "door1">}
            [dtmf-2 two] {<option "door2">}
            [dtmf-3 three] {<option "door3">}
            [(touch the monkey)] {<option "monkey">}
          ]
        ]]>
  </grammar>
```

If you compare the new grammar element above to the one on the previous page, you'll notice that we've added addition information for each option as well as a new line. The addition information consists of a single word for each option. For example, the first line of the grammar instructs the Tellme VoiceXML parser to match when the user presses 1 on their keypad or if they speak the word *one*. The last line of the grammar matches on the phrase, "touch the monkey." There will not be a match if the user says "touch" or "monkey." It will only match when the user says "touch the monkey." This is because the phrase is contained within paretheses, which require that the whole phrase be matched.

Example 2. [example2.xml]

```
<?xml version="1.0"?>
 <vxml version="1.0">
   <form>
```

```
            <field name="answer">
                    <grammar> <![CDATA[
               [
                 [dtmf-1 one] {<option "door1">}
                 [dtmf-2 two] {<option "door2">}
                 [dtmf-3 three] {<option "door3">}
                 [(touch the monkey)] {<option "monkey">}
               ]
              ]]>
                    </grammar>
                    <prompt> <audio>You are in a small room with thr
doors.</audio>
                      <pause>300</pause> <audio>To open the first do
1.</audio>
                      <pause>300</pause> <audio>To open the second d
2.</audio>
                      <pause>300</pause> <audio>To open the third do
3.</audio>
                    </prompt>
            </field>
            <filled>
                    <result name="door1">
                      <audio>You see a large hungry
monkey.</audio>
                      <reprompt/>
                    </result>
                    <result name="door2">
                      <audio>You see another room with three doors,
monkey.</audio>
                      <reprompt/>
                    </result>
                    <result name="door3">
                      <audio>You see a man scratching his
monkey.</audio>
                      <reprompt/>
                    </result>
                    <result name="monkey">
                      <audio>No! Do not touch the
monkey!</audio>
                      <reprompt/>
                    </result>
    </filled>
  </form>
</vxml>
```

Handling Events

Next, we need to add handlers for the *nomatch* and *noinput* events using the methods

specified in Page 4 [http://www.webref.com/perl/tutorial/20/4.html] of the previous tutorial.

Example 3. [example3.xml]

```
<?xml version="1.0"?>
 <vxml version="1.0">
   <form>
            <field name="answer">
                     <grammar> <![CDATA[
               [
                 [dtmf-1 one] {<option "door1">}
                 [dtmf-2 two] {<option "door2">}
                 [dtmf-3 three] {<option "door3">}
                 [(touch the monkey)] {<option "monkey">}
               ]
             ]]>
                     </grammar>
                     <prompt> <audio>You are in a small room with thr
 doors.</audio>
                       <pause>300</pause>
                       <audio>To open the first door, press or say
 1.</audio>
                       <pause>300</pause>
                       <audio>To open the second door, press or say
 2.</audio>
                       <pause>300</pause>
                       <audio>To open the third door, press or say
 3.</audio>
                     </prompt>
                     <nomatch>
                       <audio>You fool! That is not a choice I am giv
 you.</audio>
                       <pause>300</pause>
                       <audio>Press or say 1, 2, or
 3.</audio>
                       <listen/>
                     </nomatch>
                     <noinput>
                       <audio>What are you waiting
 for?</audio>
                       <pause>300</pause>
                       <audio>Press or say 1, 2, or
 3.</audio>
                       <listen/>
                     </noinput>
            </field>
            <filled>
              <result name="door1">
                       <audio>You see a large hungry
```

```
</audio>
                      <reprompt/>
                    </result>
                    <result name="door2">
                      <audio>You see another room with three doors,
 monkey.</audio>
                      <reprompt/>
                    </result>
                    <result name="door3">
                      <audio>You see a man scratching his
 monkey.</audio>
                      <reprompt/>
                    </result>
                    <result name="monkey">
                      <audio>No! Do not touch the
 monkey!</audio>
                      <reprompt/>
                    </result>
    </filled>
  </form>
 </vxml>
```

Adding Help

In most VoiceXML applications, you'll want to provide some help facility. Most people expect to get some kind of help when they press 0 on their phone, whether it be a live operator or automated assistance. In the example below, we've added a DTMF and voice grammar item that calls the *help* element in the document below.

Example 4. [example4.xml]

```
<?xml version="1.0"?>
 <vxml version="1.0">
   <form>
          <field name="answer">
                 <grammar> <![CDATA[
           [
             [dtmf-1 one first] {<option "door1">}
             [dtmf-2 two second] {<option "door2">}
             [dtmf-3 three third] {<option "door3">}
             [(touch the monkey)] {<option "monkey">}
             [dtmf-0 huh help what doh] {<option "help">}
           ]
         ]]>
                 </grammar>
                 <prompt>
                   <audio>You are in a small room with three
```

```
</audio>
                  <pause>300</pause>
                  <audio>Which one do you want to
 open?</audio>
                </prompt>
                <nomatch count="1">
                  <audio>Wrong door.</audio>
                  <audio>To open the first door, press or say
 1.</audio>
                  <pause>300</pause>
                  <audio>To open the second door, press or say
 2.</audio>
                  <pause>300</pause>
                  <audio>To open the third door,
                  press or say 3.</audio>
                  <listen/>
                </nomatch>
                <nomatch count="2">
                  <audio>You fool! That is not a choice I am giving
 you.</audio>
                  <pause>300</pause> <audio>Press or say 1, 2, or
 3.</audio>
                  <listen/>
                </nomatch>
                <nomatch count="3">
                  <audio>Pretend you have a brain and press 1, 2, o
 3.</audio>
                  <listen/>
                </nomatch>
                <noinput count="1">
                  <audio>What are you waiting
 for?</audio>
                  <pause>300</pause>
                  <audio>Press or say 1, 2, or 3.</audio>
                  <listen/>
                </noinput>
                <noinput count="2">
                  <audio>If you do not press 1, 2, or 3,
                  I am going to kill the monkey!</audio>
                  <listen/>
                </noinput>
                <noinput count="3">
                  <audio>Ok. That's it. The monkey is dead and it's
                  your fault because you did not choose 1, 2, or
 3.</audio>
                  <listen/>
                </noinput>
                <noinput count="4">
                  <audio>1, 2, or 3.</audio>
                  <listen/>
```

```
                </noinput>
                <help>
                  <audio>Why are you asking for
help?</audio>
                  <pause>300</pause>
                  <audio>Go away.</audio>
                  <reprompt/>
                </help>
          </field>
          <filled>
            <result name="door1">
                  <audio>You see a large hungry
monkey.</audio>
                  <reprompt/>
                </result>
                <result name="door2">
                  <audio>You see another room with three doors,
                  a man, and his monkey.</audio>
                  <reprompt/>
                </result>
                <result name="door3">
                  <audio>You see a man scratching his
monkey.</audio>
                  <reprompt/>
                </result>
                <result name="monkey">
                  <audio>No! Do not touch the
monkey!</audio>
                  <reprompt/>
                </result>
    </filled>
  </form>
</vxml>
```

If you hadn't noticed, I also added multiple elements for the *nomatch* and *noinput* handlers.

Making Our Adventure Dynamic

Now that we have a working VoiceXML document that provides some dynamic capabilities, it's time to tie in into a back-end Perl script that provides true dynamic capabilities. For this first run of our dynamic Perl script, we'll develop a script that dynamically generates rooms. Each room will always have three doors. Because the script is random, there are many different possible combinations. Initially, users won't be able to interact with any of the objects in the rooms, they'll only be able to open doors. In Part III, we'll add this capability.

room.pl

```
use CGI;
use strict;
my @rooms = (
              'large ball room',
              'pool hall',
              'large red barn',
              'seven eleven',
              'bedroom',
              'bathroom',
              'funeral parlor',
              'windy room with bark walls',
              'dimly lit tunnel',
              'prison cell',
              'hospital waiting room',
              'underground parking garage',
              'circular stairwell',
              'arboretum',
              'janitors closet',
              'closet',
              'subway car',
              'cubicle',
              'lighthouse tower',
              'graveyard'
            );
my @objects = (
                'full of clowns dressed in red',
                'full of smoke but no people',
                'with a dead horse on the floor',
                'with a broken slurpy machine. In the corner, a trol
                'with black curtains and something under the covers',
                'with padded walls',
                'with wood bark walls',
                'with flickering florescent lights',
                'with an oil spotted shag carpet',
                'full of mad raving lunatics',
                'with a man and his monkey',
                'full of monkeys',
                'with William Shatner singing tambourine man',
                'with a large hungry monkey',
                'full of drunken sailors',
                'full of blind dwarves',
                'full of Windows programmers',
                'with a man scratching his monkey'
              );
my $q = new CGI;
print "Content-Type: text/xml\n\n";
```

```
&prompt;
sub prompt {
my $room   = $rooms[int(rand(scalar(@rooms)))];
my $object = $objects[int(rand(scalar(@objects)))];
   print <<VXML;
<?xml version="1.0"?>
<vxml version="1.0">
  <form>
    <field name="answer">
      <grammar>
        <![CDATA[
          [
            [dtmf-1 one first] {<option "door1">}
            [dtmf-2 two second] {<option "door2">}
            [dtmf-3 three third] {<option "door3">}
            [(touch the monkey)] {<option "monkey">}
            [dtmf-0 huh help what doh] {<option "help">}
          ]
        ]]&gt;
      </grammar>
      <prompt>
        <audio>You are in a $room $object. There are three
doors.</audio>
      </prompt>
      <filled>
        <submit
next="http://www.textant.com/cgi-bin/room.pl"/>
      </filled>
      <nomatch count="1">
        <audio>Wrong door.</audio>
        <audio>To open the first door, press or say
1.</audio>
        <pause>300</pause>
        <audio>To open the second door, press or say
2.</audio>
        <pause>300</pause>
        <audio>To open the third door, press or say
3.</audio>
        <listen/>
      </nomatch>
      <nomatch count="2">
        <audio>You fool! That is not a choice I am giving
you.</audio>
        <pause>300</pause>
        <audio>Press or say 1, 2, or 3.</audio>
        <listen/>
      </nomatch>
      <nomatch count="3">
        <audio>Pretend you have a brain and press 1, 2, or
```

```
</audio>
          <listen/>
        </nomatch>
        <noinput count="1">
          <audio>You need to select a door.</audio>
          <pause>300</pause>
          <audio>Press or say 1, 2, or 3.</audio>
          <listen/>
        </noinput>
        <noinput count="2">
          <audio>If you do not press 1, 2, or 3, I am going to kill
          the monkey!</audio>
          <listen/>
        </noinput>
        <noinput count="3">
          <audio>Ok. That's it. The monkey is dead and it's your
          fault because you did not choose 1, 2, or
 3.</audio>
          <listen/>
        </noinput>
        <noinput count="4">
          <audio>1, 2, or 3.</audio>
          <listen/>
        </noinput>
        <help>
          <audio>Why are you asking for help?</audio>
          <pause>300</pause>
          <audio>Go away.</audio>
          <reprompt/>
        </help>
      </field>
  </form>
 </vxml>
 VXML
 }
```

Conclusion

I hope this column has provided a bit of nostalgia for some of you who grew up playing paper and computer adventure games like D&D, Battletech, Adventure, Zork, Leisure Suit Larry, Loom, and Space Quest. Alas, the era of adventure games seems to have passed. No longer are games with extensive plots that really draw users into an alternate universe. Games like BloodNet and Ultima brought a short resurgence, but I'm afraid we may not see another for a while. Shootemups like Quake and Half Life have all but replaced the adventure games.

What's even more depressing is seeing some of the great gaming companies like FASA, Micropose, Spectrum Holobyte, and Infocom go out of business or get bought out by larger companies who really don't care too much about making great games.

Well, that's ok, because in Part III, we're going to develop our own text adventure platform in an XML format that will work with HTML, WAP, and VoiceXML. We'll let our nostalgia run its course and then perhaps we'll look at utilizing this technology for something truly useful. But right now, it's time to play. We'll see you in Part III. Until then, I'm going to go back through those sweet memories of yesteryear.