Powered by Universal Speech Solutions LLC

UniMRCP

Asterisk

AWS Lex and Polly

Usage Guide

Revision: 1

Created: September 28, 2018

Last updated: September 28, 2018

Author: Arsen Chaloyan

Table of Contents

1 Overview

This guide describes how to utilize the Amazon Lex and Polly services with Asterisk.

Note that the Asterisk and the UniMRCP server typically reside on different hosts in a LAN, although both might be installed on the same host.

Installation of the Asterisk and the UniMRCP server with the Amazon Lex and Polly plugins is not covered in this document. Visit the corresponding web pages for more information.

http://unimrcp.org/asterisk
http://unimrcp.org/lex
http://unimrcp.org/polly

1.1 Applicable Versions

Instructions provided in this guide are applicable to the following versions.

UniMRCP Modules for Asterisk 1.5.0 and above

UniMRCP Lex Plugin 1.0.0 and above

UniMRCP Polly Plugin 1.0.0 and above

2 Generic Speech Recognition API

2.1 Overview

The module *res_speech_unimrcp.so* provides an implementation of the Generic Speech Recognition API of Asterisk, based on the UniMRCP client library.

2.2 Configuration Steps

This section outlines major configuration steps required for use of the module *res_speech_unimrcp.so* with the UniMRCP server.

Check the name of the profile, referenced in the configuration file *res-speech-unimrcp.conf*, which is located in the configuration directory of Asterisk. Use *uni2* for MRCPv2, and *uni1* for MRCPv1.

```
[general]
unimrcp-profile = uni2      ; UniMRCP MRCPv2 Server
; unimrcp-profile = uni1    ; UniMRCP MRCPv1 Server
```

Open the configuration file *unimrcpclient.xml*, located in the configuration directory of UniMRCP and specify IP addresses of the client and the server. In the following example, the Asterisk/UniMRCP client is located on 10.0.0.1 and the UniMRCP server is on 10.0.0.2.

```
<properties>
        <!--  <ip type="auto"/> -->

        <ip>10.0.0.1</ip>
        <server-ip>10.0.0.2</server-ip>
</properties>
```

2.3 Usage Examples

Lex is a conversational engine, which typically requires multiple interactions with the caller until a dialog is complete.
The following Asterisk AGI script written in Python demonstrates a basic flow applicable to an ordinary Lex chat bot, which needs to be set up separately through the AWS Console.

```
#!/usr/bin/python
import urllib2
import xml.etree.ElementTree as ET
import sys
```

```
from asterisk.agi import *

agi = AGI()

# Create speech object
agi.appexec('SpeechCreate')

# Activate speech grammar
agi.appexec('SpeechActivateGrammar',"builtin:speech/transcribe")

# Set welcome prompt
prompt = 'Welcome to Amazon Lex. How can I help you?'

# Play prompt and perform recognition in a loop until dialog is complete
run = True
while run == True:

    # Play prompt
    agi.appexec('MRCPSynth', "%s" % prompt)

    # Perform recognition (including intent detection)
    agi.appexec('SpeechBackground')

    # Check whether results are available
    status = agi.get_variable('SPEECH(results)')
    agi.verbose("got status %s" % status)

    if status == "1":

        # Get result
        result = agi.get_variable('SPEECH_TEXT(0)')
        result = urllib2.unquote(result).decode('utf8')
        result = '<instance>' + result + '</instance>'
        agi.verbose("got result %s" % result)

        # Parse content
        root = ET.fromstring(result)
        agi.verbose("got root")

        # Find message element
        message = root.find('message')
        if message != None:
            agi.verbose("got message %s" % message.text)
            prompt = "\\\"%s\\\"" % message.text

        # Find dialogstate element
        dialogstate = root.find('dialogstate')
        if dialogstate != None:
            agi.verbose("got dialogstate %s" % dialogstate.text)
            if dialogstate.text == 'ReadyForFulfillment':
                # Dialog is complete
```

```
                run = False

        # Play final prompt
        prompt = 'Thank you. See you next time.'
        agi.appexec('MRCPSynth', "%s" % prompt)

        # Destroy speech object
        agi.appexec('SpeechDestroy')
```

Assuming the provided Python script is named agi_lex.py and located in the Asterisk agi-bin directory, associate the script to an extension, for example, 701 in the Asterisk dialplan.

```
        exten => 701,1,Answer
        exten => 701,n,AGI(agi_lex.py)
```

Place a test call and follow the prompts until the dialog is complete.

Note that the script also makes use of the MRCPSynth application in order to synthesize the next prompt played to the caller.

3 Suite of UniMRCP Applications

3.1 Overview

The module *app_unimrcp.so* provides a suite of speech recognition and synthesis applications for Asterisk.

3.2 Configuration Steps

This section outlines major configuration steps required for use of the module *app_unimrcp.so* with the UniMRCP server.

Open the configuration file *mrcp.conf*, located in the configuration directory of Asterisk, and add two profiles for MRCPv2 and MRCPv1 respectively. In the following example, the Asterisk/UniMRCP client is located on 10.0.0.1 and the UniMRCP server is on 10.0.0.2.

```
[uni2]
; MRCP settings
version = 2
;
; SIP settings
server-ip = 10.0.0.2
server-port = 8060
; SIP user agent
client-ip = 10.0.0.1
client-port = 25097
sip-transport = udp
;
; RTP factory
rtp-ip = 10.0.0.1
rtp-port-min = 28000
rtp-port-max = 29000
;
; Jitter buffer settings
playout-delay = 50
max-playout-delay = 200
; RTP settings
ptime = 20
codecs = PCMU PCMA L16/96/8000 telephone-event/101/8000
; RTCP settings
rtcp = 0
```

```
[uni1]
```

```
; MRCP settings
version = 1
;
; RTSP settings
server-ip = 10.0.0.2
server-port = 1554
resource-location = media
speechsynth = speechsynthesizer
speechrecog = speechrecognizer;
;
; RTP factory
rtp-ip = 10.0.0.1
rtp-port-min = 27000
rtp-port-max = 28000
;
; Jitter buffer settings
playout-delay = 50
max-playout-delay = 200
; RTP settings
ptime = 20
codecs = PCMU PCMA L16/96/8000 telephone-event/101/8000
; RTCP settings
rtcp = 0
```

3.3 Usage Examples

MRCPRecog

The use of this application is currently limited to one interaction only, as each MRCPRecog call is performed in the scope of a new MRCP session. The application needs to be revised to allow consecutive RECOGNIZE requests be placed in the scope of the same session in order to continue the dialog with Lex.

MRCPSynth

Use the application MRCPSynth to speech synthesis.

```
[app-unimrcp-2]
exten => s,1,Answer
exten => s,2,MRCPSynth(Welcome to Asterisk,l=en-US&p=uni2)
```

In the Asterisk dialplan, associate the provided sample to an extension, for example, 802.

```
exten => 802,1,Goto(app-unimrcp-2,s,1)
```

Place a test call and listen to the synthesized message.

SynthAndRecog

The use of this application is currently limited to one interaction only, as each SynthAndRecog call is performed in the scope of a new MRCP session. The application needs to be revised to allow consecutive RECOGNIZE requests be placed in the scope of the same session in order to continue the dialog with Lex.