Faculdade de Ciências Exatas e da Engenharia
Licenciatura em Engenharia Informática
Projeto/Estágio
2022/2023

Relatório Projeto/Estágio

Sara Ornelas Teixeira

2045419

Funchal, 21 de julho de 2023

Índice

Índice de Figuras

Glossário

Antes de começar a trabalhar com o servidor Asterisk precisamos de conhecer alguns termos comuns de telefonia e do próprio sistema.

Dialplan: O dialplan é essencialmente uma linguagem de script específica para o Asterisk e uma das principais formas de instruir o Asterisk sobre como se comportar. Através do dialplan, podemos encaminhar e manipular as chamadas[1].

Codecs: Uma biblioteca de software que contém os algoritmos necessários para transformar sinais de voz analógicos para digitais e vice-versa. Exemplos: G.711 G.729 GSM [2], [3].

VOIP (Voice Over Internet Protocol)**:** É um sistema de telefone que utiliza a conexão á internet para fazer e receber chamadas[2], [4].

RTP (Real-Time Transport Protocol): É um protocolo de rede para a entrega de áudio e vídeo pela Internet. Ele foi projetado para fornecer funções de transporte para aplicações que transmitem dados em tempo real, como áudio e vídeo[5].

PBX (Private Branch Exchange): Consiste numa rede privada de telefonia, que permite aos utilizadores comunicarem entre si, são também responsáveis por transmitir chamadas para as redes externas, incluindo o *routing* e funcionalidades mais avançadas para chamadas inbound e outbound[6].

Docker: É uma plataforma aberta para desenvolvimento, envio e execução de aplicações. O Docker permite que separemos as nossas aplicações de sua infraestrutura para que você possa entregar software rapidamente. Aproveitando as vantagens das metodologias do Docker para enviar, testar e implantar código rapidamente, podemos reduzir significativamente o atraso entre escrever código e executá-lo na produção[7].

Docker Container: O Docker fornece a capacidade de empacotar e executar uma aplicação, sistemas operativos ou aplicações como o nodejs e o mongodb, entre muitas outras, num ambiente isolado chamado container. O isolamento e a segurança permitem que executemos vários containers simultaneamente [7], [8].

Docker Compose: O Compose é uma ferramenta para definir e executar aplicações Docker de vários containers. Com o Compose, usamos arquivos YAML (.yml), para configurar os serviços, manipular as variáveis de ambiente, de segurança, rede, e os ficheiros de configurações das aplicações [9], [10].

WSL (Windows Subsystem for Linux): O WSL permite que os desenvolvedores executem um ambiente GNU/Linux -- incluindo a maioria das ferramentas de linha de comando, utilitários e aplicações -- diretamente no Windows [11].

ASR (Automatic Speech Recognition): É um recurso, no Asterisk, que permite ao sistema reconhecer e interpretar palavras ou frases faladas.

TTS (Text-to-Speech): Processo de sintetizar a fala humana a partir do texto escrito[12].

URI (Uniform Resource Identifier): Consiste numa cadeia de caracteres compactos, usado para identificar um recursos na Internet[13].

1. Introdução

Este relatório tem como objetivo apresentar a investigação e projetos desenvolvidos no estágio na empresa Altice Labs, com duração de 6 meses. O objetivo principal do estágio foi a investigação e implementação, de soluções para as funcionalidades de Text-to-Speech (TTS) e Automatic Speech Recognition (ASR), numa versão mais recente do servidor Asterisk.

O estágio enquadrou-se num projeto já existente chamado Channel Management, que consiste numa solução de multicanais de comunicação para gestão de jornadas de atendimento. Atualmente esta aplicação está integrada utilizando uma versão obsoleta do Asterisk, que tem vários problemas que foram resolvidos nas versões posteriores. Diante disso, o objetivo principal do estágio consistiu em investigar como manter as funcionalidades já existentes do projeto, e implementar novas funcionalidades, disponibilizadas pelo Asterisk.

1.1 Objetivos

O objetivo principal deste estágio é o de integrar o Plugin WMS diretamente com o Asterisk, substituindo assim o WMS e permitindo evoluir para uma versão recente do Asterisk.

Uma vez que está em curso um outro estágio com o objetivo de permitir ao WMS integrar com uma versão recente do Asterisk através de um Wrapper desenvolvido em Golang, acredita-se que desse estágio seja possível criar um componente que o Plugin WMS possa integrar como um módulo Golang e assim a integração (através deste novo componente) seja facilitada.

Tendo por base o pressuposto anterior, numa fase inicial deste estágio, pretende-se que sejam investigadas soluções que se possam adotar para incorporar as funcionalidades de TTS, ASR e utilização de outras Apps disponibilizadas pelo Asterisk.

Propõe-se que seja criada uma aplicação demo, com as capacidades atualmente disponíveis pelo PluginWMS (Call Control, PlayAnnouncement) para se entender a mecânica de serviços IVR. De seguida investigar a forma de dar a funcionalidade TTS, quer seja pelo desenvolvimento de uma aplicação própria ou utilização de outra open source.

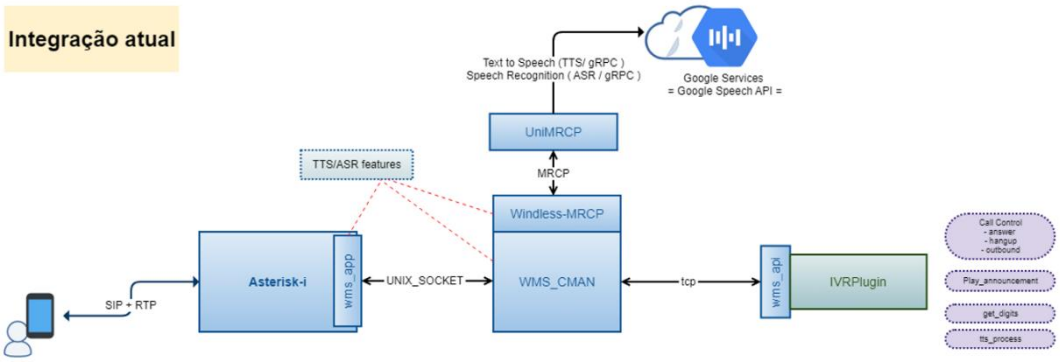

Figura 1: Integração atual da aplicação Channel Management

Durante este estágio enquadrei-me na equipa responsável por uma solução de multicanais de comunicação para gestão de jornadas de atendimento intitulada por Channel Management, onde existe uma aplicação em desenvolvimento, a Figura 1 mostra a implementação atual que utiliza uma versão do servidor Asterisk-i, uma versão modificada com base na versão 1.0 do servidor para fornecer os serviços necessários como o Text-to-Speech, o play announcements que é o toque de anúncios pré-gravados, entre outras.

O uso desta versão antiquada gera alguns problemas já que muitas das funcionalidades que foram acrescentadas ao Asterisk-i pela equipa de desenvolvimento da Altice Labs, foram disponibilizadas nas versões posteriores junto com a correção de bugs, problemas e limitações, por isso estive encarregue de investigar como fazer algumas destas funcionalidades numa versão mais recente nomeadamente, a capacidade de Text to Speech (TTS) e a de Automatic Speech Recognition (ASR).

A aplicação do Channel management consiste na criação de jornadas parametrizadas de comunicação, que são atribuídas a um número telefónico, os usos mais comuns desta aplicação são em linhas de atendimento, consultórios, e serviços que diariamente recebem imensas chamadas e que precisam de algo para dar o tratamento necessário, como por exemplo encaminhamento para atendimento por humano ou para um Bot com reconhecimento de fala.

Por exemplo quando é recebida uma chamada, pela interface da aplicação podemos decidir o que vai acontecer, podemos escolher tocar um anúncio de bem-vindas, tocar uma música de espera, colocar o cliente numa fila de espera.

Todas as informações e configurações da jornada são guardadas num ficheiro JSON que depois terá de ser guardado na base de dados, e a comunicação feita com os serviços TTS/ASR é feita pelo módulo UniMRCP.

O WMS é o que vai processar a chamada que depois irá passar este evento ao IVRPlugin através do socket WMS-CMAN, e depois este evento é passado para o Flow Engine que irá aplicar a lógica do projeto nesta chamada.

2. Solução

Nesta seção temos algumas propostas de solução para o problema da implementação das funcionalidades TTS e ASR, no contexto global da aplicação do Channel Management. Uma das propostas é o desenvolvimento de um pacote em Golang dentro do Plugin WMS que permitirá a integração do IVRPlugin, e a outra é a criação de um wrapper que vai envolver o IVRPlugin e o WMS, tendo a comunicação feita com o servidor através da interface ARI. Existe um outro estágio a decorrer sobre estas duas propostas, o meu estágio está focado na investigação das duas funcionalidades, ver Figura 4, que é a proposta de solução para este estágio.

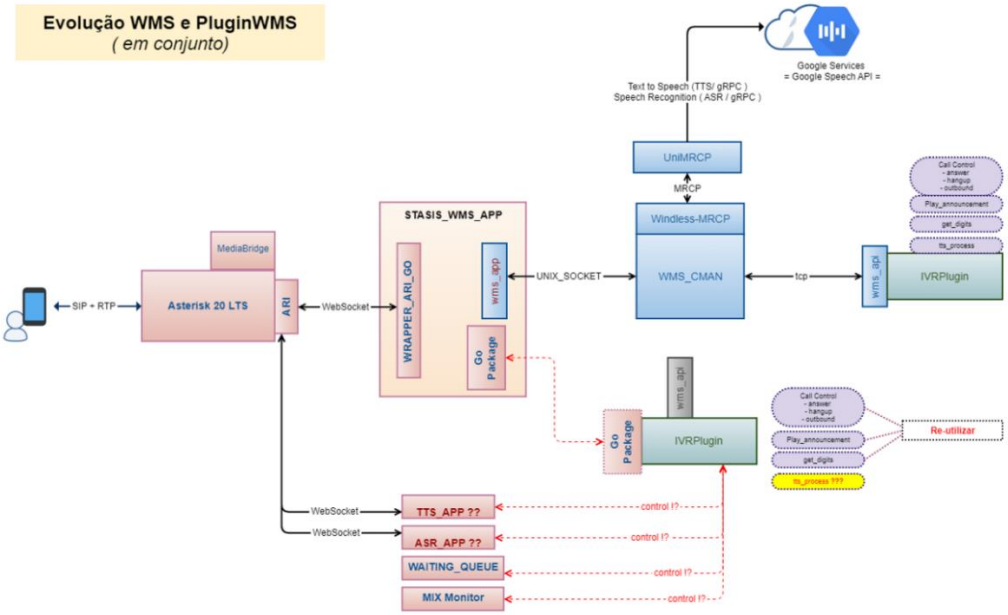

Figura 2: Proposta de solução: Evolução WMS e PluginWMS em conjunto

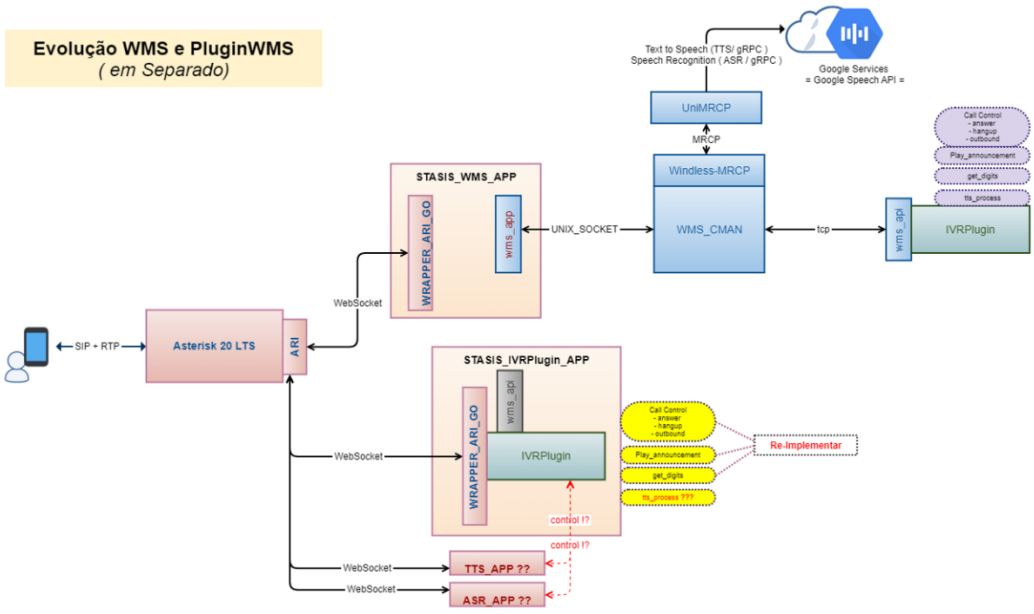

Figura 3: Proposta de solução: Evolução WMS e PluginWMS em separado

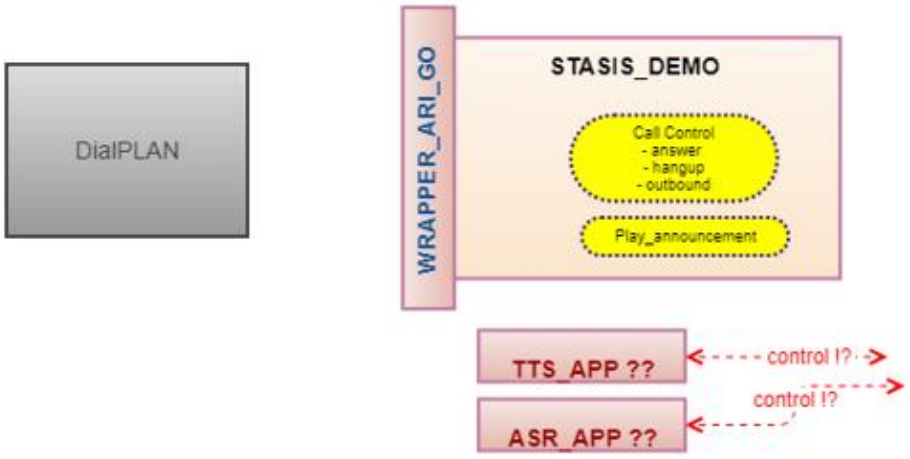

Figura 4: Proposta de solução: Investigação das funcionalidades TTS e ASR

3. Revisão de leitura

Dois dos principais conceitos para o desenvolvimento deste projeto são os conceitos de IVR e o protocolo SIP. Nesta secção ambos serão apresentados em detalhe. Além disso apresentamos também outras ferramentas tecnológicas cruciais para o desenvolvimento da solução proposta.

3.1 IVR (Interactive Voice Response)

O IVR consiste num sistema telefônico automatizado que combina mensagens pré-gravadas ou tecnologias de sintetização de áudio tais como Text-to-Speech de maneira a permitir que os utilizadores realizem chamadas e naveguem dentro das mesmas, através de uma interface de multifrequência de tom duplo (DTMF) para enviar mensagens. Permitindo que os utilizadores acedam ou forneçam informação, sendo direcionados para a extensão desejada. Por exemplo, ao pressionar 1 para Vendas ou 2 para Suporte, o sistema encaminha a chamada para essa extensão[2], [14], [15].

Se o sistema IVR não puder recuperar as informações que o chamador está procurando, as opções de menu programadas podem fornecer assistência no encaminhamento de chamadas para o representante apropriado para obter ajuda. Ao integrar as tecnologias de computação e telefonia, o software IVR pode melhorar o fluxo de chamadas e reduzir o tempo de espera, levando a uma maior satisfação geral do cliente[15].

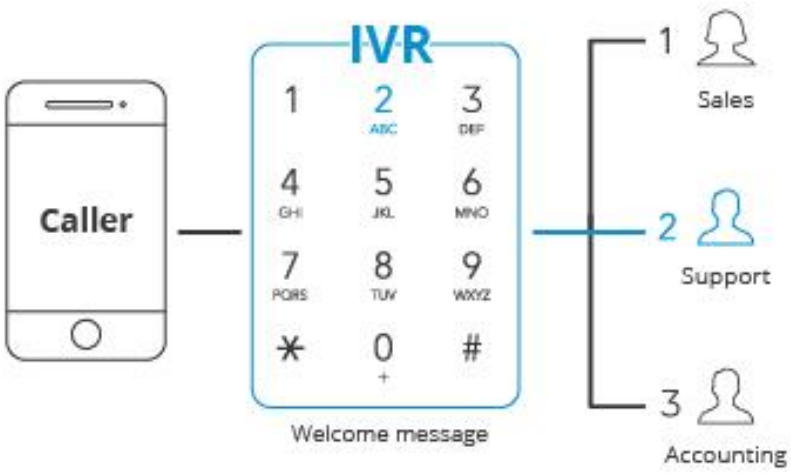

Figura 5: Representação gráfica de um sistema IVR

3.2 SIP (Session Initiation Protocol)

O SIP é um protocolo de sinalização, usado para controlar sessões de comunicação multimédia, uma sessão SIP é qualquer tipo de comunicação ao vivo, chamadas de telefone, conferências ou até de vídeo chamadas[2], [16].

O protocolo SIP funciona com comunicação bidirecional. Por cada mensagem SIP, um dispositivo envia um request, e o outro recebe e envia uma resposta. As respostas são codificadas baseadas na sua mensagem, e isto é importante para depois encontrarmos erros ao analisarmos estes

pacotes. Para o transporte de pacotes de dados podemos utilizar o Transmission Control Protocol (TCP) ou o User Datagram Protocol (UDP).

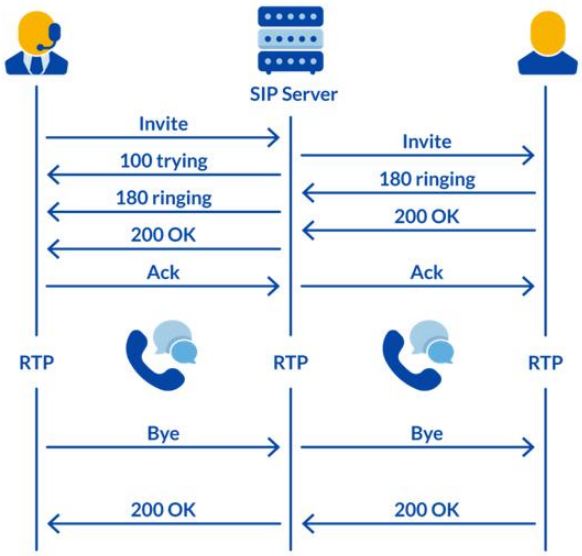

Figura 6: Chamada VoIP estabelecida com o Session Initiation Protocol (SIP)

Se quisermos uma comunicação mais exata usamos o TCP já que o envio de pacote e dados tem sempre de receber uma resposta da confirmação da receção, caso o recetor não tenha enviado essa mensagem, esse pacote será reenviado.

Se quisermos uma comunicação mais rápida, usamos o UDP já que ele não precisa da confirmação da receção do pacote enviado, é preferível usarmos este protocolo quando uma comunicação constante é necessária.

3.3 Servidor Asterisk

O Asterisk começou por uma necessidade que o seu criador Mark Spencer teve quando começou um negócio de suporte comercial ao sistema operativo Linux, em 1999, chamado 'Linux Support Services', que agora é chamado de Digium. O LSS oferecia uma linha direta de suporte para a qual os profissionais de IT podiam ligar para obter ajuda com o Linux.

Em poucos meses, o crescimento do negócio exigiu um sistema telefônico que pudesse distribuir chamadas uniformemente pela equipe de suporte, então Mark ligou para vários fornecedores de sistemas telefônicos locais e pediu orçamentos, só que todos tinham preço demasiados exorbitantes. Em vez de ceder e fazer um empréstimo para uma pequena empresa, ele decidiu desenvolver o seu próprio sistema telefônico.

Em apenas alguns meses, Mark criou o código central original do Asterisk. Assim que teve um protótipo funcionando, publicou o código-fonte na Internet, disponibilizando-o sob a licença GPL (a mesma licença usada para Linux)[17].

O Asterisk começou como um sistema telefônico, um 'open-source PBX', para uma pequena empresa, mas desde que foi lançado originalmente, ele se tornou uma ferramenta universal para criar aplicações de comunicação. Atualmente ele passou também a suportar sistemas IP

PBX, mas como também VOIP gateways, call centers, conference bridges, servidores de voice-mail e qualquer tipo de aplicações que envolvam comunicações em tempo real[18].

3.3.1 Funcionamento

O Asterisk é para as aplicações de comunicação o que o servidor web Apache é para as aplicações web. Apache é um servidor web. Asterisk é um servidor de comunicação. O Apache lida com todos os detalhes de baixo nível de envio e recebimento de dados usando o protocolo HTTP. O Asterisk lida com todos os detalhes de baixo nível de envio e recebimento de dados usando vários protocolos de comunicação diferentes.

Quando instalamos o Apache, temos um servidor web, mas é a nossa responsabilidade criar as aplicações webs. Quando instalamos o Asterisk, temos um servidor de comunicação, mas é a nossa responsabilidade criar as aplicações de comunicação.

Aplicações web são construídas a partir de páginas HTML, folhas de estilo CSS, scripts de processamento do lado do servidor, imagens, bancos de dados, serviços web, etc. Para que uma aplicação web funcione, você precisa do servidor da web conectado à Internet. Para que uma aplicação de comunicação funcione, você precisa do servidor de comunicação conectado aos serviços de comunicação (VoIP ou PSTN). Para que as pessoas possam aceder um website é preciso registar um nome de domínio e configurar entradas de DNS que apontem "www.yourdomain.com" para o servidor. Para que as pessoas acedam o nosso sistema de comunicações, precisamos de números de telefone ou URIs de VoIP que enviem chamadas para o nosso servidor.

Aqui está um exemplo simples. Este script HTML, correndo sobre um servidor Web, mostra 'Hello World' em letras grandes numa página Web:

```
<html>
<head>
<title>Hello World Demo</title>
</head>
<body>
<h1>Hello World! </h1>
</body>
</html>
```

Este script do Dialplan a seguir atende o telefone, espera um segundo, reproduz um áudio 'Hello World' e desliga.

```
exten => 100,1,Answer()
exten => 100,n,Wait(1)
exten => 100,n,Playback(hello-world)
exten => 100,n,Hangup()
```

Em ambos os casos, os componentes do servidor estão lidando com todos os detalhes de baixo nível. Não precisamos de nos preocupar com o alinhamento de bytes, o tamanho do pacote, os codecs, para isso serve o Asterisk[18].

3.3.2 Protocolos e Interfaces

Existem muitas maneiras de utilizar interfaces com o Asterisk com scripts, outras aplicações ou sistemas de armazenamento. Desde o mais básico, como usar arquivos de chamada do Asterisk, até APIs mais complexas, como a ARI (Asterisk Rest Interface). Esta seção irá abranger algumas das interfaces integradas do Asterisk relevantes ao problema em questão, todas as quais fornecem algum aspeto de controle, monitoração ou armazenamento[19].

3.3.2.1 Asterisk Manager Interface

O Asterisk Manager Interface (AMI) permite que um programa se conecte a uma instância do Asterisk e que emita comandos ou que leia eventos através de um fluxo TCP/IP. Isto é particularmente útil para tentar rastrear o estado de um cliente dentro do Asterisk e direcionar esse cliente com base em regras personalizadas[20].

AMI destina-se a funções de tipo de gerenciamento. O gerenciador é um modelo cliente/servidor sobre TCP. Com o AMI podemos controlar o PBX, originar chamadas, monitorizar *channels* e *queues*, bem como executar comandos sob o Asterisk.[19]

Consiste num simples protocolo baseado em linha "key: value" que é utilizado para a comunicação entre o cliente conectado e o servidor[20].

O protocolo tem as seguintes características:

- Antes de emitir comandos para o Asterisk, temos de estabelecer uma *manager session*.
- Os pacotes podem ser transmitidos em qualquer direção a qualquer momento após a autenticação.
- A primeira linha de um pacote terá uma chave de "**Action**" quando enviada do cliente para o Asterisk, mas "**Event**" ou "**Response**" quando enviada do Asterisk para o cliente.
- A ordem das linhas dentro de um pacote é insignificante, então podemos o dicionário não ordenado nativo de qualquer linguagem de programação, para guardar eficientemente num único pacote.
- CR/LF é usado para delimitar cada linha e uma linha em branco (dois CR/LF em uma linha) indica o final do comando que o Asterisk agora deve processar [20].

3.3.2.2 Asterisk Gateway Interface

A AGI fornece uma interface entre o dialplan do Asterisk e um programa externo (via pipes, stdin e stdout), também podemos considerar como uma interface que adiciona funcionalidade ao Asterisk de maneira fácil e eficiente, utilizando várias linguagens de programação tais como Perl, PHP, C, Pascal, Phyton [19], [21].

No geral, a interface é síncrona, espera até que o request feito acabe para depois continuar a execução.[22] O uso da AGI possibilita a realização de tarefas fora das capacidades normais do Asterisk sem nenhuma dificuldade[21].

Canais de comunicação como STDIN, STDOUT e STDERR são usados na comunicação entre os scripts AGI e o Asterisk, normalmente os scripts AGI são guardados numa localização especifica **/var/lib/asterisk/agi-bin**.

STDIN – entrada, usada para comunicar dados à interface externa.

STDOUT – saída, usada para comunicar dados de volta ao Asterisk.

STDERR – erro, canal de comunicação para mensagens de erro

Durante a pesquisa por soluções TTS é de notar que a maioria utilizava a interface AGI, e em relação ao ASR se queremos ter uma implementação onde não tenhamos de desenvolver nenhum módulo é preferível usar a AGI já que ela tem controlo sobre o dialplan e as aplicações de dialplan no que se refere às aplicações relacionadas com Speech.

A diferença entre a AMI e a AGI é que a AMI é considerado como third party control e não aparece no dialplan, enquanto a AGI é considerado como first party control e tem de sempre aparecer no dialplan[23].

3.3.2.3 Asterisk REST Interface

ARI é uma API assíncrona que permite aos desenvolvedores criar aplicações de comunicação expondo os objetos primitivos no Asterisk - channels, bridges, terminais, média, etc. - por meio de uma interface RES. O estado dos objetos controlados pelo utilizador são transmitido por meio de eventos JSON através de um WebSocket[19].

Enquanto a AMI seja boa para o controle de chamadas e a AGI seja boa em permitir que um processo remoto execute aplicações do dialplan, nenhuma dessas APIs foi projetada para permitir que um desenvolvedor crie a sua própria aplicação de comunicação[24].

Estes recursos eram tradicionalmente da responsabilidade dos módulos C do Asterisk. Ao entregar o controle desses recursos a todos os desenvolvedores, o Asterisk se torna um mecanismo de comunicação, onde a lógica de negócios de como as coisas se devem comunicar fica á responsabilidade da aplicação que está a usar o Asterisk.

ARI is not about telling a channel to execute the Voicemail dialplan application or redirecting a channel in the dialplan to Voicemail.

It is about letting you build your own Voicemail application. [24]

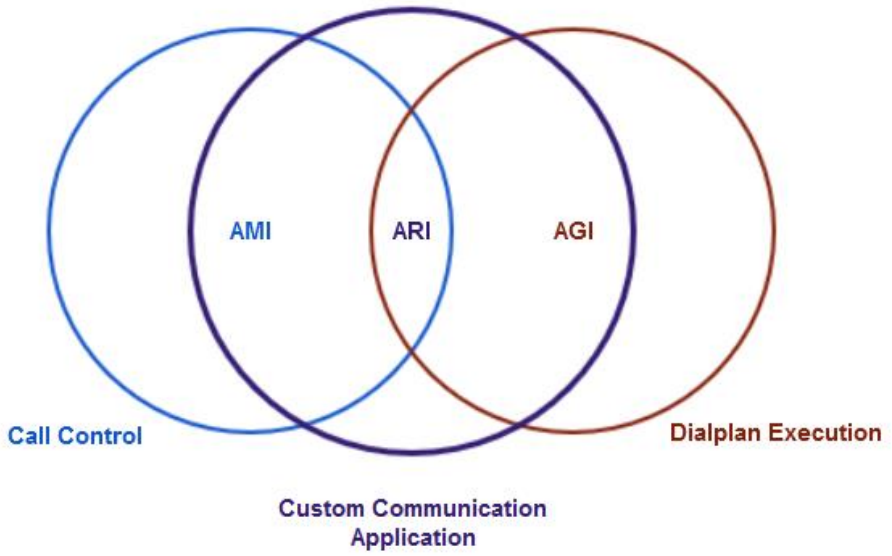

Figura 7: Relação entre AMI, ARI e AGI.

A ARI consiste em três tecnologias que são - para todos os efeitos - inter-relacionadas e usadas juntas. Elas são:

- Uma interface RESTful que o cliente utiliza para controlar os recursos do Asterisk.
- Uma WebSocket que transmite eventos em JSON sobre os recursos do Asterisk para o cliente.
- A aplicação do dialplan chamada Stasis, que entrega o controlo de um canal do Asterisk para o cliente.

Todas as três partes trabalham juntas, permitindo que um desenvolvedor manipule e controle os recursos fundamentais do Asterisk e construa a seu própria aplicação de comunicação[24].

REST

O Representational State Transfer (REST) é uma arquitetura de software, com as seguintes características:

- A comunicação é realizada utilizando um modelo cliente-servidor.
- A comunicação é sem estado. Os servidores não armazenam o estado do cliente entre as solicitações.
- É uma interface uniforme. Os recursos são identificados nos requests, as mensagens são auto-descritivas, etc.

ARI não está estritamente em conformidade com a API REST. O Asterisk, como uma aplicação autônomo, possui um estado que pode mudar fora de uma solicitação do cliente por meio do ARI. Por exemplo, um telefone SIP pode ser desligado e o Asterisk desligará o canal - mesmo que um cliente através do ARI não tenha dito ao Asterisk para desligar o telefone SIP[24].

WebSocket

WebSockets são um padrão de protocolo relativamente novo (RFC 6455) que permite a comunicação bidirecional entre um cliente e um servidor. O objetivo principal do protocolo é fornecer um mecanismo para aplicações *browser-based* que precisam de comunicação bidirecional com os servidores, sem depender de pesquisas longas de HTTP ou outros mecanismos não padrões.

No caso do ARI, uma conexão WebSocket é usada para passar eventos assíncronos do Asterisk para o cliente. Esses eventos estão relacionados à interface RESTful, mas são tecnicamente independentes dela. Eles permitem que o Asterisk informe ao cliente as mudanças no estado do recurso que podem ocorrer devido e em conjunto com as mudanças feitas pelo cliente através do ARI[24].

Stasis

Stasis é uma aplicação de dialplan no Asterisk. É o mecanismo que o Asterisk usa para entregar o controlo de um canal do dialplan - que é a forma tradicional de controlar os canais - para a ARI e o cliente. Geralmente, as aplicações ARI manipulam canais na aplicação do dialplan Stasis, bem como outros recursos no Asterisk. Os canais que não estão em numa aplicação do dialplan Stasis geralmente não podem ser manipulados pelo ARI - o objetivo do ARI, afinal, é construir a sua aplicação do dialplan, não manipular um existente[24].

3.3.2.4 Asterisk External Application Protocol

O Asterisk External Application Protocol (AEAP) é uma framework desenvolvida para facilitar o desenvolvimento de módulos no Asterisk que precisam de comunicar com aplicações externas[25]. Atualmente, é usado para implementar um mecanismo de Speech-to-Text que se conecta a uma aplicação externa [17].

Esta interface é usada para comunicar configurações, dados, e outros tipos de informações usando um sistema de mensagens de request/response. Atualmente só suporta JSON para as mensagens[25].

A API AEAP está localizada nos arquivos res_aeap.h e res_aeap_message.h, estes contêm todos os métodos necessários para se conectar, receber/enviar mensagens e ler/gravar áudio de/para uma aplicação remota[26].

O Asterisk External Application Protocol (AEAP) é realmente duas coisas, dependendo da perspetiva. Do ponto de vista do desenvolvedor do Asterisk, é uma API e um protocolo usado para conectar e comunicar com uma aplicação externa ao Asterisk.

3.3.2.5 Speech Recognition API

A Speech Recognition API do dialplan é baseada num arquivo de aplicações de vários métodos, funcionalidades relacionadas com a fala, que exportam muitas aplicações a serem usados para reconhecimento de fala. Isto inclui uma aplicação para preparar o reconhecimento de fala, ativar a gramática e reproduzir um áudio enquanto espera a pessoa falar. Usando uma combinação destas aplicações, podemos facilmente fazer um dialplan para usar o reconhecimento de fala, sem nos preocuparmos com qual mecanismo de reconhecimento de fala está sendo usado[19].

3.4 Ferramentas utilizadas

Durante o estágio, existiu a oportunidade de utilizar várias ferramentas novas e que chegaram a ser essenciais para o desenvolvimento deste trabalho. Nesta seção, serão descritas as principais ferramentas que foram utilizadas.

O Sngrep é uma ferramenta utilizada para mostrar os pacotes SIP apanhados em tempo real [27]. Esta ferramenta foi bastante importante quando comecei a utilizar o servidor Asterisk, porque pude monitorizar as comunicações feitas e analisar o tráfego de pacotes e assim consegui observar o que estava a acontecer quando fazia uma chamada, foi especialmente útil para encontrar problemas de comunicação com o servidor.

Além do Sngrep, também utilizei o Wireshark, uma ferramenta de análise de protocolos de rede[28]. Utilizei o Wireshark quando tive a necessidade de ver outros pacotes além de SIP, por exemplo quando testei a aplicação AEAP Speech-to-Text.

Outra ferramenta que desempenhou um papel importante tanto na organização como na pesquisa foi o GitHub. Utilizei o GitHub para pesquisar por projetos e códigos relacionados ao meu trabalho, aproveitando a vasta comunidade e os recursos disponíveis. Além disso, utilizei o GitHub como um repositório para armazenar as minhas implementações.

Para o ambiente de desenvolvimento, optei por utilizar o Ubuntu WSL (Windows Subsystem for Linux). Escolhi usar o WSL e não uma máquina virtual pela facilidade de uso e eficiência, e também pela sugestão do meu orientador de estágio.

Em relação às chamadas, utilizei o SIP Phoner Lite, apesar da configuração inicial tenha sido um pouco mais complicada em comparação a outros programas utilizados, tais como o Zoiper Classic e o Zoiper 5, descobri que o SIP Phoner Lite oferecia uma eficiência superior e não apresentava problemas significativos ao contrário dessas duas ferramentas que quando utilizei encontrei vários problemas de conexão ao servidor.

Estas ferramentas desempenharam um papel significativo no meu estágio, permitindo que eu adquirisse experiência prática com redes, com conseguir detetar e solucionar problemas de conexão, e que pudesse seguir com o desenvolvimento do estágio de uma maneira mais eficaz.

4. Proposta de Solução

Nesta seção, serão apresentadas propostas de soluções para os seguintes problemas, a instanciação do servidor Asterisk e da implementação das funcionalidades TTS e ASR.

4.1 Instanciação do Servidor Asterisk

A primeira tarefa que tive que fazer durante este estágio foi ter de instanciar o Asterisk dentro de um container do Docker, e como ponto de partida comecei por utilizar uma aplicação chamada Docker Desktop [29].

Apesar desta aplicação ter uma interface bem organizada e de fácil uso, foi bastante complicado encontrar uma imagem do Asterisk que funcione corretamente, e quando encontrei uma não soube como manusear o container nem que configurações alterar, e por esta mesma razão fiquei presa neste ponto, até que pela orientação do meu orientador de estágio experimentei usar o WSL e instalar o sistema operativo Ubuntu 18.04.

A instalação do WSL [11], [30] foi simples, através desta abordagem tive um maior controlo sobre os container e de melhorar os meus conhecimentos de sistemas operativos *nix. Durante o estágio houve alguns momentos em que o sistema quebrou e ficou corrompido, mas pelo facto do WSL ter uma comunidade enorme, facilitou a correção de bugs e os vários erros que aconteceram.

Com um ambiente de desenvolvimento, novamente pela sugestão do meu orientador de estágio, fiz a instalação desta imagem Docker do Asterisk, pelo GitHub, chamada mlan/docker-asterisk [31]. Esta imagem foi sugerida pela vasta documentação fornecida e pelo guia de instalação e de uso[32], que foi explícito e fácil de seguir e foi uma referência importante para ter uma instância do servidor Asterisk a funcionar.

Concluindo as etapas anteriores, consegui implementar com sucesso um sistema funcional do Asterisk. Para garantir a disponibilidade e a segurança dos ficheiros de configurações, criei um diretório de backup dentro da pasta de instalação do Asterisk, e alterei o ficheiro docker-compose.yml de maneira que tivesse acesso aos ficheiros que estão dentro do container e os guardar nesta pasta de backup. Isto permitiu a realização de backups regulares dos arquivos e configurações do Asterisk, protegendo contra perda de dados e facilitando a recuperação em caso de falhas.

4.1.1 Asterisk Hello World

Após ter uma instância do Asterisk com todos os ficheiros de configurações necessários, foi necessário testar se de facto o servidor está a funcionar fiz o que é normal um programador fazer, um Hello World.

Para fazer um Hello World segui o exemplo da wiki do Asterisk [33]. Segui todas as instruções alterando todos os ficheiros mencionados nesta página, mas, contudo, tive de alterar o ficheiro pjsip_transport.conf alterando as primeiras 5 configurações colocando o IP do meu servidor.

Com estes passos concluídos tentei fazer uma chamada inicialmente com a aplicação Zoiper Classic, mas não consegui ouvir o áudio, tentei novamente com outra aplicação Zoiper 5, mas novamente sem sucesso. Para tentar entender o que estava a acontecer utilizei uma ferramenta de observação de pacotes SIP, chamada sngrep.

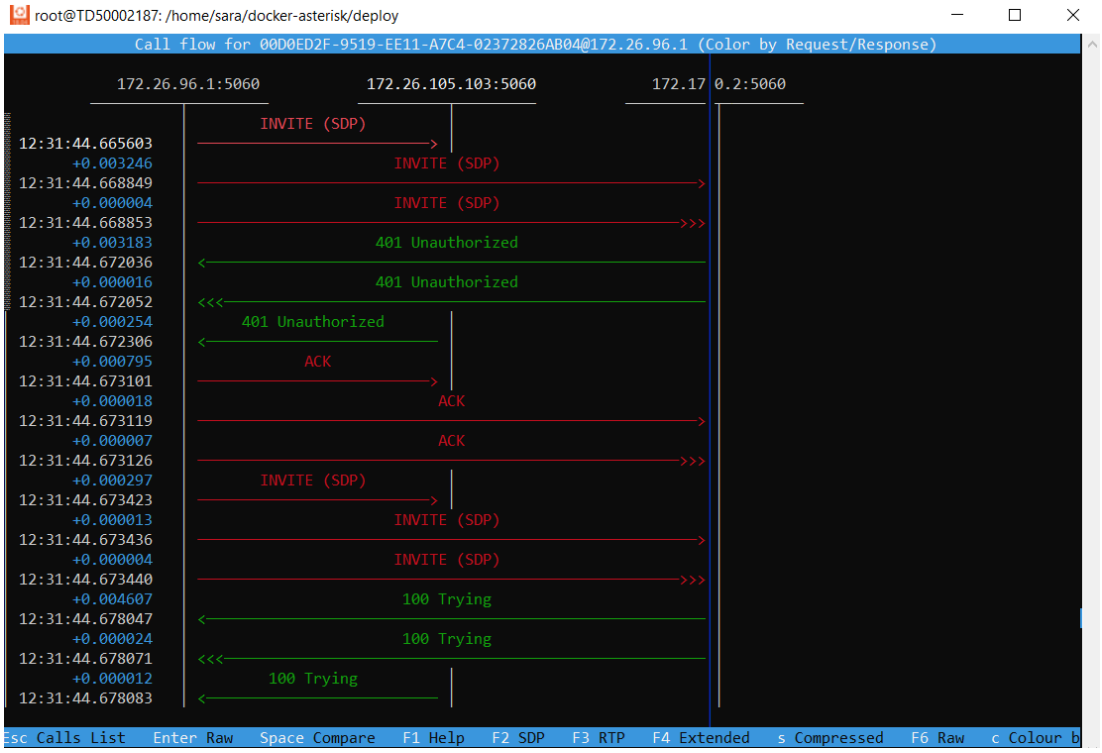

Figura 8: Chamada 'Hello World' com insucesso. Observação com sngrep

Podemos observar que na Figura 7 aparecem 3 endereços de IP distintos, o primeiro da esquerda para a direita, era o da minha máquina por onde a chamada, o segundo endereço era o do sistema operativo Ubuntu, e o último endereço é o endereço *default* do Docker [34]. Isto significa que a chamada não estava a chegar ao Asterisk porque ao fazer a chamada ela ia para o IP do Docker e ficava por aí. Para resolver isto, alterei a configuração de rede dentro do docker-compose.yml de bridge, a configuração que veio com a instalação, para o modo host, dizendo ao Docker que se adapte ao IP do Ubuntu e não use o endereço *default* [10], [34], e mudei da aplicação para fazer as chamadas para utilizar o SIP Phoner Lite.

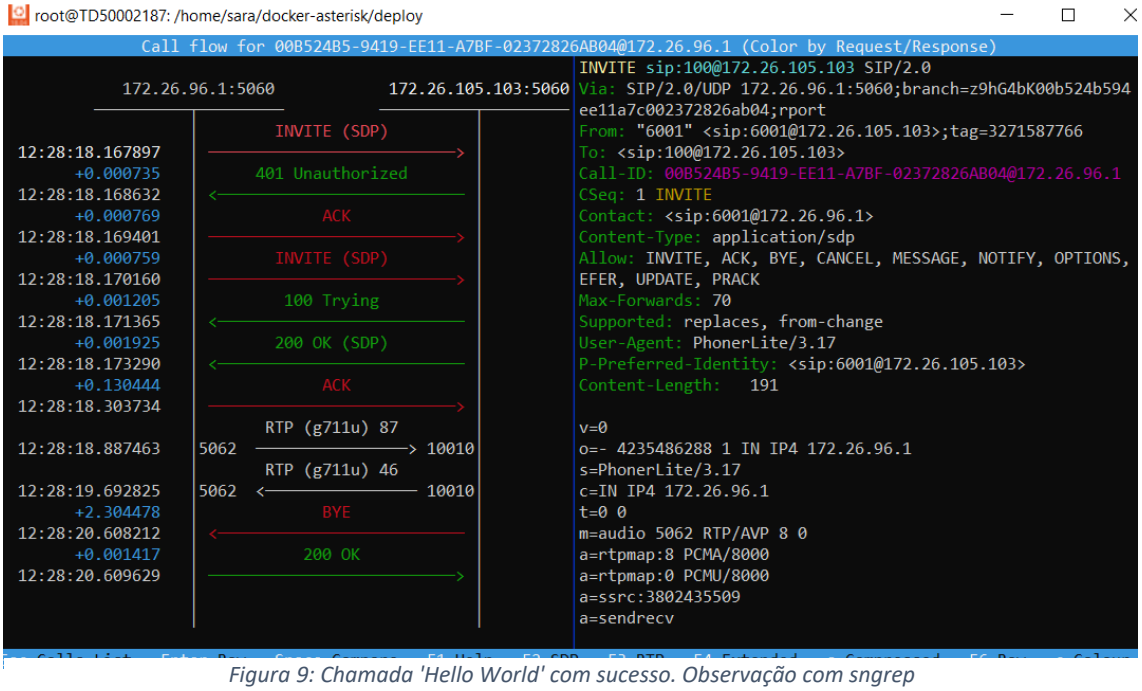

Figura 9: Chamada 'Hello World' com sucesso. Observação com sngrep

Na Figura 8, após com as alterações mencionadas posteriormente, já podemos observar as diferenças entre a chamada que falhou e uma que sucedeu, ver Figura 6. Temos agora 2 endereços IP e podemos ver a comunicação a ser feita com troca de mensagens, e vemos também o protocolo RTP em ação, que é o áudio 'Hello World' a ser projetado.

4.2 Text-to-Speech

A solução que queremos chegar tem de ser algo modular, algo que não esteja agarrado ao Asterisk, por exemplo se quisermos mudar o *engine* utilizado para a sintetização de áudio, as alterações feitas têm de ser minimais.

A implementação da funcionalidade é baseada num exemplo já disponibilizado [35] pelo repositório CyCoreSystems, que toca um som através de uma funcionalidade Play que toca sons por URI. Então sabendo isto a ideia da implementação atual, é fazer a sintetização de um texto e tocar o ficheiro de áudio gerado, através da função Play. Contudo, veremos na seção de implementação que só com isto, não irá funcionar, teremos de alterar algumas configurações no request de TTS e no como é que o servidor vai ter acesso a este ficheiro de áudio.

Para além da pesquisa sobre tecnologias e soluções do Asterisk que possam ser utilizadas, foram realizadas pesquisas para encontrar soluções existentes que pudessem ser utilizadas para a funcionalidade de TTS. O critério para viabilidade dessas soluções incluía a presença de documentação explícita, licença de uso adequada ao contexto comercial e um desenvolvimento ativo.

Foram encontradas diversas soluções que atendem aos critérios estabelecidos, incluindo soluções oferecidas pela Google, soluções específicas para o Asterisk e outras opções baseadas em bibliotecas de terceiros, contudo não foram testadas.

Estas são as soluções encontradas Asterisk-TTS[36], Asterisk Assistant[37], gTTS[38], asterisk-googletts[39], precisam ainda ser exploradas. De notar que a maioria destas aplicações utilizam a interface AGI e a linguagem Phyton para implementar a funcionalidade de TTTS.

4.2.1 External Media

A partir da versão Asterisk 16.6 apresenta um novo método para permitir a interação com um servidor de média externo. Usando o novo recurso ARI "/channels/externalMedia", um desenvolvedor pode direcionar a média para a sua própria aplicação, que por sua vez, poderá reencaminhar para um engine de sintetização, como a Google, para análise[40].

Na figura seguinte, é apresentado um cenário a nossa aplicação ARI cria um novo canal de External media, fornecendo alguns parâmetros básicos, como o destino e formato da média, e adiciona esse canal a uma *bridge* existente. O channel driver encaminha todas as médias da bridge para o destino. A aplicação ARI pega nos dados e coloca-os no formato que correspondem aos requisitos do engine de reconhecimento de voz escolhido

O canal de External Media também pode injetar média numa bridge da qual é membro para que seja possível reproduzir mensagens de progresso, música, menus IVR, etc. Por esta razão, esta solução é a mais recomendada, pela grande capacidade de manipular a média e o envio dessa mesma média para os engines de sintetização.

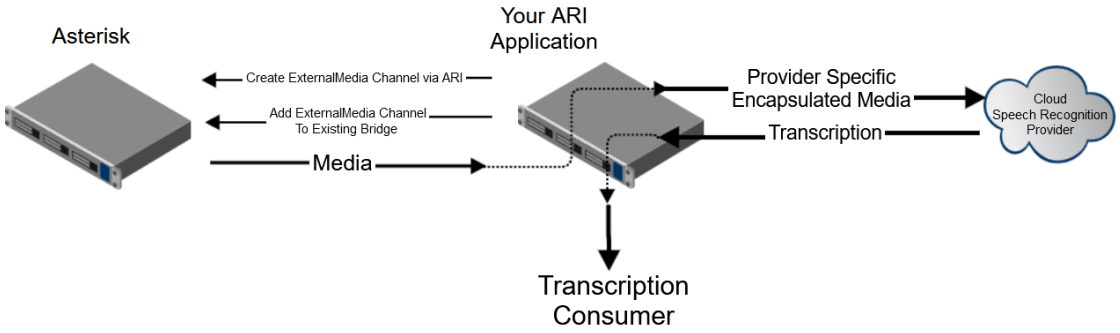

Figura 10 : Proposta de solução: ARI com o uso do External Media

O pacote Asterisk-external-media demonstra como usar o recurso ARI External Media para transcrever o áudio de uma bridge usando as APIs de Speech do Google [41].

4.3 Automatic Speech Recognition

Inicialmente, foi indicado a utilização de um outro pacote da ARI chamado ari-proxy[42], que teoricamente suportaria a execução de aplicações do dialplan pela ARI, permitindo assim utilizar as funções do Speech Recognition API[35]. No entanto, ao testar e examinar os métodos disponíveis no pacote ari-proxy que podem ser executados no objeto Chanel, não encontrei nada. A ideia era fazer essa execução através do Channel.Execute(), mas desta maneira executar as funções speech é impossível.

A segunda abordagem consistiu em implementar o ASR da mesma forma que o Text-to-Speech, ou seja, utilizando a ARI para tratar com todo o processo de sintetização. Ao fazer isto não estamos a aproveitar os recursos do Asterisk, e implicaria que a solução tenha os mesmos problemas que a implementação do TTS, mas se seguirmos isto teremos algo funcional apesar de não ser o ideal.

Após estas tentativas iniciais, decidi explorar outras possibilidades antes de prosseguir com a segunda abordagem. Durante esta pesquisa adicional, descobri como utilizar uma aplicação do dialplan chamada TALK_DETECT [36], que permite detetar silêncios e a presença da fala num canal de comunicação.

Com base nisso, foi realizada uma análise para verificar se seria possível executar as funções do Speech Recognition API. Como parte desse processo, verifiquei os fóruns da comunidade e coloquei algumas perguntas sobre este assunto, mas de facto não é possível utilizar as aplicações speech pela ARI.

'*The speech ones can't really be used from ARI. They control the dialplan applications or exist as a result of them, and dialplan applications are not executable in ARI. You would have to send the channel to the dialplan.*' – Jcolp, Asterisk Project Lead

Diante das limitações das abordagens anteriores, decidi seguir a segunda abordagem, porém com uma nova perspetiva baseada numa solução proposta para o mesmo problema [38] encontrado no fórum da comunidade do Asterisk. A ideia consistia em utilizar os métodos de

gravação (record) do repositório CyCoreSystems/ari para gravar o áudio e, posteriormente, realizar o reconhecimento de fala utilizando a biblioteca speech da Google [39].

4.3.1 Aplicação Speech-to-Text

A aplicação **aeap-Speech-to-Text** foi desenvolvida pelos desenvolvedores do Asterisk, e utiliza o protocolo AEAP e um tem desenvolvimento ativo, utiliza a licença Apache License 2.0. Permite o uso comercial, modificações e distribuição[43].

A teoria por detrás desta aplicação é a seguinte, ela será responsável por agarrar a média e determinar o que fazer com ela com base nas informações fornecidas pelo Asterisk. Os dados são enviados de um lado para o outro por uma conexão de WebSocket, na forma de JSON. A aplicação externa depois encaminhará a média para um serviço de fala, como Google ou Amazon. A partir daí, o resultado será passado de volta para o Asterisk.

Este programa age como um intermediário entre o Asterisk e o Google Speech Provider, criando um servidor WebSocket que fica á espera de conexões dos clientes através do Asterisk. Quando uma conexão for estabelecida, e houver uma comunicação com o protocolo AEAP então o *speech audio* pode ser enviado pelo Asterisk para a aplicação, que depois vai enviar para os serviços da Google. Após receber o Speech-to-Text da Google, o módulo vai enviar os resultados para o Asterisk através de uma mensagem AEAP[43].

Antes de qualquer outra solução mencionada na seção anterior, a intenção seria utilizar este protocolo e esta aplicação para fazer o reconhecimento de fala. O problema acontece na falta de conhecimentos de como integrar esta aplicação desenvolvida em Java, com a ARI desenvolvida em Golang, e como esta aplicação precisa de estar sempre a correr junto ao Asterisk significa que temos de ter duas aplicações a correrem simultaneamente. Se quisermos uma solução modular, seguir este caminho não nos levará a isso, por isso o uso desta aplicação foi mais para teste e relevante para a investigação e aprendizagem.

5. Implementação

Na seção 5, avançamos para a fase de implementação das funcionalidades TTS e ASR, seguindo as abordagens e soluções propostas na seção 4, será desenvolvido o funcionamento geral e algumas dificuldades encontradas.

5.1 Text-to-Speech

Após pesquisar sobre como fazer uma implementação da funcionalidade Text-to-Speech, obtive a indicação de utilizar como base um exemplo Play[35] do repositório CyCoreSystems/ari [24] que toca um ficheiro de som através da ARI, utilizando a linguagem Golang.

Antes de tudo é importante colocar o exemplo a funcionar para termos uma base que sabemos que de facto funciona, e para isso segui o README do exemplo Play[35] mas com a imagem utilizada tive de fazer algumas alterações no ficheiro modules.conf para ativar todos os módulos da ARI.

Esta implementação consiste em duas partes, uma versão modificada do exemplo Play e um servidor que pega no ficheiro de áudio gerado pelo TTS e faz upload desse ficheiro para o endereço local, de maneira que o Asterisk possa ter acesso a este ficheiro por request HTTP.

```
        // Toca o áudio gerado pela função textToSpeech.
        if err := play.Play(ctx, h,
play.URI("sound:http://192.168.1.84:5000/sound/output.wav")).Err();
err != nil {
                log.Error("falha ao tocar o som", "erro", err)
                return
        }
```

Supostamente se usarmos localhost deveria funcionar, mas ao testar observamos que não é possível, essa é a razão que no excerto de código aparece um endereço IP. Com isto feito aconteceu outro problema, o Asterisk não conseguia ter acesso ao ficheiro de áudio, mas existia uma incompatibilidade entre o ficheiro e o que o Asterisk conseguia processar.

Para fazer a sintetização de áudio utilizamos um excerto de código em Golang da biblioteca de cliente da Text-to-Speech[44], consiste numa função que utiliza a API TTS da Google Cloud Platform que gera um arquivo de áudio no formato mp3 a partir do texto 'Hello World'. É criado um cliente para se ligar á API, envia um request com várias configurações com o texto a ser sintetizado, com as configurações de voz, de linguagem, e do formato do ficheiro gerado. Depois recebemos o ficheiro de áudio gerado como resposta.

Para resolver o problema de compatibilidade, tive de testar os vários formatos de áudio disponíveis, corria o programa e observava os erros que Asterisk dava, e tentava corrigir. As configurações funcionais são as seguintes:

```
      // Select the type of audio file you want returned.
      AudioConfig: &texttospeechpb.AudioConfig{
         AudioEncoding:   texttospeechpb.AudioEncoding_LINEAR16,
         SampleRateHertz: 8000, // Set the sample rate to 8000
Hz
      },
```

5.2 Automatic Speech Recognition

Tendo como base a implementação anterior, e com a abordagem definida, vamos gravar todo o áudio do canal, isto inclui áudio que não era suposto ser gravado, utilizando os métodos record do repositório CyCoreSystems/ari [24].

```
res, err := record.Record(ctx, h,
   record.IfExists("overwrite"),
   record.MaxDuration(3*time.Second),
).Result()
```

O áudio gravado vai ter a duração de 3 segundos, após isso na variável res vamos ter acesso a esta gravação, e temos disponível o áudio URI da gravação e podemos tocar no canal.

```
   if err := play.Play(ctx, h, play.URI(res.URI())).Err(); err
!= nil {
      log.Error("failed to play sound", "error", err)
      return
   }
```

O código utilizado para fazer o reconhecimento utiliza a API Google Cloud Speech para transcrever áudio. Criamos um cliente para se comunicar com a API e, em seguida, envia um request para transcrever o áudio. No exemplo base, o áudio é localizado no URI "gs://cloud-samples-data/speech/brooklyn_bridge.raw", por isso a ideia de utilizar o `res.URI()`. O cliente recebe a resposta da API contendo o texto transcrito, e depois o resultado é impresso na consola.

6. Avaliação

A implementação da funcionalidade Text-to-Speech pela ARI, apesar de ser algo simples e funcional tem alguns problemas óbvios e por isso não é recomendado utilizar esta solução num contexto comercial. Os problemas em questão são os seguintes:

A dependência de um servidor externo introduz uma fraqueza adicional no sistema, se o servidor de áudio estiver inacessível ou enfrentar problemas de conectividade, poderá acontecer um atraso entre o request HTTP e a reprodução do áudio na chamada ou no pior dos casos a reprodução do áudio falhar completamente.

Ao fazer um request HTTP para buscar os ficheiros de áudio, para que tenhamos uma aplicação segura temos de implementar algumas medidas de segurança, o que nesta implementação não está feito, como a autenticação e encriptação para proteger os dados e evitar acessos não autorizados aos ficheiros de áudio.

Dependendo do número de chamadas simultâneas e do tamanho dos ficheiros de áudio, o servidor de som irá encontrar problemas de escalabilidade, esta solução é adequada para um projeto pequeno, por isso para o contexto do estágio esta solução deve ser descartada.

A implementação da funcionalidade de ASR não está completamente funcional. Existem alguns problemas de compatibilidade com o ficheiro URI gerado pela gravação e o reconhecimento de áudio do STT da Google. Para além disto existem os seguintes problemas a serem considerados:

O problema mais importante é o da qualidade do áudio gravado pode afetar a precisão do reconhecimento de fala, se houver demasiado barulho no background da chamada é possível que nem seja possível fazer o reconhecimento, e se fizer talvez seja pouco preciso e incompleto.

Alguns serviços de Speech-to-Text podem ter custos associados ao uso, especialmente em termos do volume de áudio processado. Por isto utilizar as aplicações de reconhecimento de fala do Asterisk seria o melhor, porque ao gravar a chamada podemos gravar áudio desnecessário, e a elevar o custo desnecessariamente.

7. Discussão

Relacionando o problema inicial e a proposta deste estágio sobre a investigação de soluções para as funcionalidades TTS e ASR, podemos concluir que é possível implementar estas funcionalidades. Temos vários meios de chegar a uma solução funcional e que se irá enquadrar no projeto Channel Management. Durante a investigação de soluções para ambas as funcionalidades, exploramos diferentes caminhos e tecnologias disponíveis no ecossistema do Asterisk, buscando a melhor abordagem para atender às necessidades do sistema.

Para o TTS utilizar a implementação proposta não é a mais recomendada, mas identificamos uma alternativa viável através do uso do External Media. Que oferece uma maneira eficaz de manipular os dados presentes no canal de comunicação. Esta abordagem permitirá a síntese de voz em tempo real, garantindo que o sistema possa fornecer informações audíveis de forma dinâmica e natural para os clientes.

Em relação ao Automatic Speech Recognition (ASR), não foi possível chegar a uma implementação funcional utilizando a ARI. Para esta funcionalidade é recomendado o uso da AGI (Asterisk Gateway Interface) para aproveitar os recursos disponibilizados pelo próprio Asterisk, permitindo a integração com mecanismos de reconhecimento de fala já estabelecidos.

8. Conclusão

Este relatório apresenta várias propostas de soluções documentadas e com as referências necessárias para a progressão da investigação, com a implementação das soluções mais viáveis tendo em conta a documentação, e as minhas capacidades.

Apesar das implementações serem falhas em termos de segurança e viabilidade, exploram soluções alternativas que exploram tecnologias novas e comprovam que de facto a implementação destas funcionalidades é possível.

Futuramente se dado a oportunidade, a implementação do External Media é algo essencial, porque para além de ser a solução mais ideal ela utiliza várias tecnologias e conceitos essenciais para o projeto.

9. Referências

[1] «Asterisk Dialplan». https://wiki.asterisk.org/wiki/display/AST/Dialplan
[2] «Asterisk Glossary». https://www.asterisk.org/get-started/glossary/
[3] «What Are VoIP Codecs & How Do They Affect Call Sound Quality?» https://www.nextiva.com/blog/voip-codecs.html
[4] «What Is VoIP & How Does It Work?» https://www.nextiva.com/blog/what-is-voip.html
[5] «What is the Real-time Transport Protocol (RTP)?» https://www.tutorialspoint.com/real-time-transport-protocol-rtp
[6] «What is a PBX? The Definitive Guide to Private Branch Exchange Systems». https://www.nextiva.com/blog/what-is-pbx.html
[7] «Docker overview | Docker Documentation». https://docs.docker.com/get-started/overview/
[8] «What is a Container? - Docker». https://www.docker.com/resources/what-container/
[9] «Docker Compose overview | Docker Documentation», *Docker Compose overview | Docker Documentation*. https://docs.docker.com/compose/features-uses/
[10] «Networking in Compose | Docker Documentation». https://docs.docker.com/compose/networking/
[11] «What is the Windows Subsystem for Linux?» https://learn.microsoft.com/en-us/windows/wsl/about
[12] «Você sabe o que é TTS?» https://medium.com/dialograma/voc%C3%AA-sabe-o-que-%C3%A9-tts-dfa2e3835c6b
[13] «URI». https://pt.wikipedia.org/wiki/URI
[14] «What is Touch Tone?» https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjayIzDsPX_AhUaVqQEHdFUC0EQFnoECBYQAQ&url=https%3A%2F%2Fwww.easytechjunkie.com%2Fwhat-is-touch-tone.htm&usg=AOvVaw272eo7tiwL-fXuFGkqE5t3&opi=89978449
[15] «What is interactive voice response?» https://www.ibm.com/topics/interactive-voice-response
[16] «What Is Session Initiation Protocol (SIP) & How Does It Work?» https://www.nextiva.com/blog/sip-protocol.html
[17] «A Brief History of the Asterisk Project». https://wiki.asterisk.org/wiki/display/AST/A+Brief+History+of+the+Asterisk+Project
[18] «Asterisk as a Swiss Army Knife of Telephony». https://wiki.asterisk.org/wiki/display/AST/Asterisk+as+a+Swiss+Army+Knife+of+Telephony
[19] «Interfaces - Asterisk Project». https://wiki.asterisk.org/wiki/display/AST/Interfaces
[20] «AMI | Asterisk Manager API». https://www.voip-info.org/asterisk-manager-api/
[21] «Asterisk AGI: Asterisk Gateway Interface». https://www.voip-info.org/asterisk-agi/
[22] «Asterisk Gateway Interface (AGI)». https://wiki.asterisk.org/wiki/pages/viewpage.action?pageId=32375589
[23] «What's the difference between AGI and AMI». https://community.asterisk.org/t/whats-the-difference-between-agi-and-ami/26738/4
[24] «Golang Asterisk REST Interface (ARI) library». [Em linha]. Disponível em: https://github.com/CyCoreSystems/ari
[25] «Asterisk External Application Protocol (AEAP)». https://wiki.asterisk.org/wiki/pages/viewpage.action?pageId=47875006
[26] «Asterisk External Application Protocol: The Framework». https://www.asterisk.org/asterisk-external-application-protocol-the-framework/
[27] irontec, «Sngrep | Ncurses SIP Message Flow Viewer». [Em linha]. Disponível em: https://github.com/irontec/sngrep

[28] «Wireshark | Documentation». [Em linha]. Disponível em: https://www.wireshark.org/docs/
[29] «Install Docker Desktop on Windows». [Em linha]. Disponível em: https://docs.docker.com/desktop/install/windows-install/
[30] «Install Linux on Windows with WSL». [Em linha]. Disponível em: https://learn.microsoft.com/en-us/windows/wsl/install
[31] mlan, «The mlan/asterisk repository». [Em linha]. Disponível em: https://github.com/mlan/docker-asterisk
[32] «How to run Asterisk in docker container». https://computingforgeeks.com/how-to-run-asterisk-in-docker-container/
[33] R. Newton, «Hello World - Asterisk Wiki». https://wiki.asterisk.org/wiki/display/AST/Hello+World
[34] «How to change the default docker subnet IP range». https://support.hyperglance.com/knowledge/changing-the-default-docker-subnet
[35] «CyCoreSystems / ari || Examples - Play». [Em linha]. Disponível em: https://github.com/CyCoreSystems/ari/blob/master/_examples/play/main.go
[36] «mcilley / Asterisk-TTS». [Em linha]. Disponível em: https://github.com/mcilley/Asterisk-TTS
[37] «bkbilly / asterisk-assistant». [Em linha]. Disponível em: https://github.com/bkbilly/asterisk-assistant
[38] «pndurette / gTTS». [Em linha]. Disponível em: https://github.com/pndurette/gTTS
[39] «zaf / asterisk-googletts». [Em linha]. Disponível em: https://github.com/zaf/asterisk-googletts
[40] «External Media and ARI». https://wiki.asterisk.org/wiki/display/AST/External+Media+and+ARI
[41] «asterisk / asterisk-external-media». [Em linha]. Disponível em: https://github.com/asterisk/asterisk-external-media
[42] «CyCoreSystems / ari-proxy». [Em linha]. Disponível em: https://github.com/CyCoreSystems/ari-proxy
[43] «asterisk / aeap-speech-to-text». [Em linha]. Disponível em: https://github.com/asterisk/aeap-speech-to-text
[44] «Bibliotecas de cliente da Text-to-Speech». [Em linha]. Disponível em: https://cloud.google.com/text-to-speech/docs/libraries?hl=pt-br
[45] «The Asterisk Speech Recognition API». https://wiki.asterisk.org/wiki/display/AST/Speech+Recognition+API
[46] «Docker image providing Asterisk PBX». [Em linha]. Disponível em: https://github.com/mlan/docker-asterisk