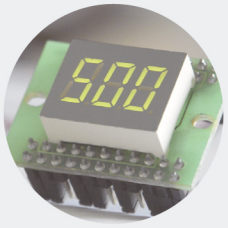
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Habilitação de Reconhecimento de Fala num Servidor de Media e Controlo Avançado de Chamadas

GONÇALO DA ROCHA ARAÚJO
Julho de 2024

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Media Server Speech Enabling and Advanced Call Control

Gonçalo Araújo

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Automação e Sistemas

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Julho, 2024

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Automação e Sistemas.

Candidato: Gonçalo Araújo, Nº 1190615, 1190615@isep.ipp.pt

Orientação Científica: Paula Viana, pmv@isep.ipp.pt

Empresa: Altice Labs, S.A

Orientador: Virgílio Cunha, virgilio-a-cunha@alticelabs.com

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Julho, 2024

Esforço, Dedicação, Devoção e Glória

Agradecimentos

Gostaria de aproveitar esta oportunidade para expressar a minha gratidão a todas as pessoas que contribuíram para a conclusão do meu projeto de mestrado.

Ao meu orientador, Virgílio Cunha, o meu sincero reconhecimento pela orientação e por ter disponibilizado o seu tempo para partilhar o seu conhecimento, pela ajuda nos momentos mais difíceis e, acima de tudo, pela confiança e amizade.

À minha orientadora, Professora Paula Viana, o meu profundo agradecimento pelo apoio constante, pelas valiosas sugestões e pela orientação ao longo deste percurso.

À equipa da Altice Labs, o meu reconhecimento pela colaboração e apoio ao longo deste projeto.

Mais importante ainda, devo a minha mais profunda gratidão à minha família pelo apoio, compreensão e amor, sem os quais a conclusão desta tese não teria sido possível.

Resumo

O *Windless Media Server* (WMS) é uma solução desenvolvida pela Altice Labs para serviços de multimédia, tais como toque de anúncios áudio, gravação de chamadas e criação de conferências. Os serviços multimédia do WMS são servidos pela aplicação *open-source* Asterisk, sobre o protocolo de sinalização *Session Initiation Protocol* (SIP) em conjunto com os protocolos *Session Description Protocol* (SDP) e *Real-Time Transport Protocol* (RTP). De forma a acompanhar a evolução tecnológica do Asterisk, houve também a necessidade de evoluir a integração do WMS com o Asterisk. Esta integração passou a ser feita através de uma aplicação denominada `WMS Adapter` que tira partido das *Application Programming Interface*s (APIs) que o Asterisk disponibiliza. A aplicação responsável por fazer a interligação entre o WMS e o Asterisk encontra-se em fase de protótipo, pelo que apenas suporta recursos básicos de controlo de chamadas. Contudo, no que toca a recursos de *Automatic Speech Recognition* (ASR) e *Text-To-Speech* (TTS), não existe suporte nativo dado pelo Asterisk, sendo que atualmente estas funcionalidades são dadas por outras componentes do servidor de multimédia.

Pretende-se com este projeto dar continuidade ao desenvolvimento do protótipo, implementando funcionalidades em falta, incluindo a incorporação de recursos de reconhecimento e síntese de fala através do Asterisk.

Palavras-Chave: WMS, TTS, ASR, Asterisk, `WMS Adapter`, SIP, SDP, RTP, Asterisk APIs, Ambientes Conteinerizados.

Abstract

The *Windless Media Server* (WMS) is a solution developed by Altice Labs for multimedia services, such as audio announcement playback, call recording and conference creation. WMS multimedia services are served by the open-source Asterisk application, using the *Session Initiation Protocol* (SIP) along with *Session Description Protocol* (SDP) and *Real-Time Transport Protocol* (RTP). In order to keep up with the technological evolution of Asterisk, there was a need to enhance the integration of WMS with Asterisk. This integration now utilizes a application called `WMS Adapter` that leverages the *Application Programming Interfaces* (APIs) provided by Asterisk. The application responsible for bridging WMS and Asterisk is currently in the prototype phase, supporting only basic call control features. However, when it comes to ASR and TTS resources, there is no native support provided by Asterisk, and these functionalities are currently handled by other components of the media server.

This project aims to continue the development of the prototype by implementing the missing functionalities, including the incorporation of speech recognition and synthesis resources through Asterisk.

Keywords: WMS, TTS, ASR, Asterisk, `WMS Adapter`, SIP, SDP, RTP, Asterisk APIs, Containerized Environments.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Lista de Acrónimos

AEAP	*Asterisk External Application Protocol*
AGI	*Asterisk Gateway Interface*
AMI	*Asterisk Manager Interface*
API	*Application Programming Interface*
ARI	*Asterisk RESTful Interface*
ASR	*Automatic Speech Recognition*
CMAN	*Channel Management*
CNAME	*Canonical Name*
CSRC	*Contributing Source*
DTMF	*Dual-Tone Multi-Frequency*
HTTP	*Hypertext Transfer Protocol*
IA	*Inteligência Artificial*
IAX	*Inter-Asterisk eXchange*
IETF	*Internet Engineering Task Force*
IMS	*IP Multimédia Subsystem*
IP	*Internet Protocol*
IVR	*Interactive Voice Response*
JSON	*JavaScript Object Notation*
MRCP	*Media Resource Control Protocol*
MRF	*Media Resource Function*
MRS	*Media Resource Server*
NAT	*Network Address Translation*

PBX IP *Private Branch Exchange over Internet Protocol*

PHP *Hypertext Preprocessor*

PJSIP *PJ Session Initiation Protocol*

RFC *Request for Comments*

RR *Receiver Report*

RTCP *Real-Time Transport Control Protocol*

RTP *Real-Time Transport Protocol*

S-CSCF *Serving-Call Session Control Function*

SCTP *Stream Control Transmission Protocol*

SDES *Source Description Items*

SDP *Session Description Protocol*

SIP *Session Initiation Protocol*

SIP AS *Aplication Server SIP*

SMS *Short Message Service*

SR *Sender Report*

SSRC *Synchronization Source*

TCP *Transmission Control Protocol*

TRS *Telephony Resource Server*

TTS *Text-To-Speech*

UA *User Agent*

UAC *User Agent Client*

UAS *User Agent Server*

UDP *User Datagram Protocol*

URI *Uniform Resource Identifier*

URL *Uniform Resource Locator*

VoIP *Voice Over Internet Protocol*

VXML *VoiceXML*

WMS *Windless Media Server*

XML *Extensible Markup Language*

Capítulo 1

Introdução

A evolução tecnológica na área das telecomunicações tem impulsionado a necessidade de desenvolver soluções mais avançadas e integradas para a gestão de serviços multimédia. O Asterisk, uma plataforma de código aberto amplamente utilizada para *Private Branch Exchange over Internet Protocol* (PBX IP) e comunicação *Voice Over Internet Protocol* (VoIP) [1], tem sido uma ferramenta essencial nesta área. Contudo, a sua constante evolução apresenta desafios na manutenção e evolução de aplicações que dependem das suas funcionalidades.

O *Windless Media Server* (WMS), desenvolvido pela Altice Labs, é uma solução projetada especificamente para fornecer serviços multimédia avançados, como reprodução de anúncios de áudio, reconhecimento automático de fala, síntese de texto para fala, gravação de chamadas, criação de conferências, entre outros. Este sistema depende das funcionalidades do Asterisk para operar de maneira eficiente e integrada.

1.1 Definição do Problema

Originalmente, o Asterisk não fornecia os recursos avançados necessários à evolução tecnológica pretendida do WMS, o que exigiu que esses requisitos fossem desenvolvidos internamente. Como solução, foi criada uma versão modificada do Asterisk, denominada Asterisk-i, para incorporar funcionalidades adicionais e recursos específicos. Para alcançar tais objetivos, houve a necessidade de desenvolvimento no *core* do próprio Asterisk, sendo que esta abordagem acabou por tornar a migração

para novas versões do Asterisk cada vez mais complexa, implicando muito esforço por parte da equipa de desenvolvimento para garantir o correto funcionamento das implementações previamente realizadas.

Por forma a mitigar este problema, foi desenvolvida uma aplicação, denominada `WMS Adapter`, que integra com o Asterisk através da *Asterisk RESTful Interface* (ARI), e com o WMS por meio de um *socket* UNIX, mantendo o protocolo de comunicação previamente usado com o Asterisk-i, tornando a migração do WMS para esta nova arquitetura transparente. Esta abordagem de integração do `WMS Adapter` com o Asterisk traz mais flexibilidade e facilidade para migrações futuras do mesmo.

Este projeto pretende incorporar funcionalidades avançadas de controlo de chamadas na aplicação já existente do `WMS Adapter`, incluindo suporte a *Automatic Speech Recognition* (ASR) e *Text-To-Speech* (TTS), possibilitar a utilização de aplicações *Dialplan* fornecidas pelo Asterisk e adaptar o `WMS Adapter` para ambientes conteinerizados.

Para este projeto, vários objetivos específicos podem ser delineados. Estes são:

- **Implementação de Chamadas de Saída:** Desenvolver funcionalidades para iniciar e gerir chamadas de saída no `WMS Adapter`.
- **Implementação de *Route Endpoint*:** Implementar a capacidade de roteamento de chamadas através de *endpoints*.
- **Integração com Aplicações *Dialplan* do Asterisk:** Desenvolver a funcionalidade de invocação de aplicações de *Dialplan* do Asterisk através do `WMS Adapter`.
- **Integração de ASR e TTS com o `WMS Adapter`:** Fornecer o acesso a recursos de reconhecimento automático de fala e síntese de texto para fala através do `WMS Adapter`.
- **Conteinerização do `WMS Adapter` e do Asterisk:** Configurar e implementar a execução do `WMS Adapter` e do Asterisk em ambientes conteinerizados.

1.2 Apresentação da Entidade

A Altice Labs, antiga PT Inovação, é uma empresa de inovação tecnológica com sede em Portugal. Foi criada em 2016 pela Altice Portugal, uma empresa de telecomunicações com sede no mesmo país. A Altice Labs tem como objetivo desenvolver soluções inovadoras para o mercado das telecomunicações, com foco em tecnologias emergentes como a *Real-Time Transport Control Protocol* (RTCP), *Inteligência Artificial* (IA) e 5G [2].

A Altice Labs tem uma equipa de cerca de 1.000 colaboradores, distribuídos por escritórios em Portugal e no Brasil [3]. A empresa tem uma forte presença internacional, e os seus produtos e serviços são utilizados por clientes em todo o mundo.

1.3 Organização da Dissertação

Este documento está estruturado da seguinte forma:

- **Capítulo 1:**

  Este capítulo apresenta a definição do problema, objetivos, resultados esperados, plano de trabalho, e a entidade envolvida no projeto.

- **Capítulo 2:**

  Este capítulo explora os principais conceitos e tecnologias utilizados no projeto, com foco especial no Asterisk e suas funcionalidades, arquitetura e *Application Programming Interface*s (APIs), além de incluir uma revisão detalhada sobre o WMS e seu papel na integração com o Asterisk.

- **Capítulo 3:**

  Neste capítulo é descrita a evolução do `WMS Adapter` e a implementação das chamadas de saída e *route endpoints*, além da integração com aplicações em `Go`.

- **Capítulo 4:**

  Este capítulo apresenta as abordagens para invocar aplicações *Dialplan* do Asterisk, assim como a implementação da abordagem mais adequada.

- **Capítulo 5:**

  Neste capítulo são exploradas e implementadas as abordagens para integração de ASR e TTS no `WMS Adapter`.

- **Capítulo 6:**

  O capítulo final oferece uma conclusão sobre o trabalho realizado e sugere direções para desenvolvimentos futuros.

Capítulo 2

Estado de Arte

Este capítulo pretende apresentar os principais conceitos e tecnologias utilizados na execução do projeto.

2.1 Asterisk

O Asterisk [4], também reconhecido pelo símbolo asterisco (*), é um *software* de código aberto usado na implementação de servidores de comunicação telefónica como VoIP e sistemas PBX IP [1]. Idealizado em 1999 por Mark Spencer e com capacidade de operar em diversos sistemas operativos como o *Microsoft Windows*, *Linux* e *Mac OS X*, o projeto Asterisk destaca-se pela colaboração internacional de desenvolvedores, promovendo melhorias contínuas e inovações no domínio das comunicações [5].

2.1.1 Principais Funcionalidades

Para além da tecnologia VoIP e do sistema PBX IP, o Asterisk oferece uma ampla gama de funcionalidades [5], tais como:

- **Conferências:** Permite a criação de salas de conferência para comunicação multiutilizador.
- ***Interactive Voice Response* (IVR):** Apresenta um sistema interativo que automatiza interações no atendimento ao cliente, capacitando os utilizadores de interagir através de teclas de toque [6].

- **Gravação de Chamadas:** Permite a gravação das chamadas realizadas.
- **Encaminhamento Avançado de Chamadas:** Fornece recursos flexíveis de encaminhamento de chamadas, permitindo personalização com base em diversas condições.
- **Suporte a Múltiplos *Codecs* de Áudio:** Suporta uma variedade de *codecs* de áudio para garantir a qualidade e a interoperabilidade de chamadas.
- **Suporte a Protocolos de Comunicação:** Integra-se com diversos protocolos como *Session Initiation Protocol* (SIP) e *Inter-Asterisk eXchange* (IAX).
- **Mensagens de Voz e Correio de Voz:** Oferece funcionalidades de mensagens e correio de voz para uma gestão mais eficaz.
- **Integração com Aplicações Externas:** Permite a integração com aplicações e serviços externos por meio de APIs.
- **Suporte a Videochamadas:** Oferece suporte a videochamadas, permitindo a comunicação por vídeo além das tradicionais chamadas de voz.
- **Flexibilidade e Customização:** Sendo uma plataforma de código aberto, o Asterisk é altamente flexível e pode ser personalizado para atender a diferentes necessidades.

Estas características tornam o Asterisk uma escolha versátil e robusta para implementações de sistemas de telefonia e comunicação empresarial.

2.1.2 Arquitetura do Sistema

O *core* do Asterisk tem a capacidade de interagir com uma variedade de módulos. Estes módulos disponibilizam canais que seguem as diretrizes de encaminhamento do Asterisk para executar comportamentos programados e simplificar a comunicação entre dispositivos ou programas externos ao Asterisk [7], tal como demonstra a Figura 2.1.

Podemos dividir a arquitetura do Asterisk nos seguintes componentes:

Dialplan

O *Dialplan* desempenha um papel central e estratégico no funcionamento do Asterisk, sendo responsável por definir as diretrizes de encaminhamento e tratamento das chamadas telefónicas. Este componente é configurado mediante a utilização de ficheiros de configuração ou, em determinados contextos, através da manipulação direta de uma base de dados. Na secção 2.1.3 iremos abordar com mais detalhe este tema.

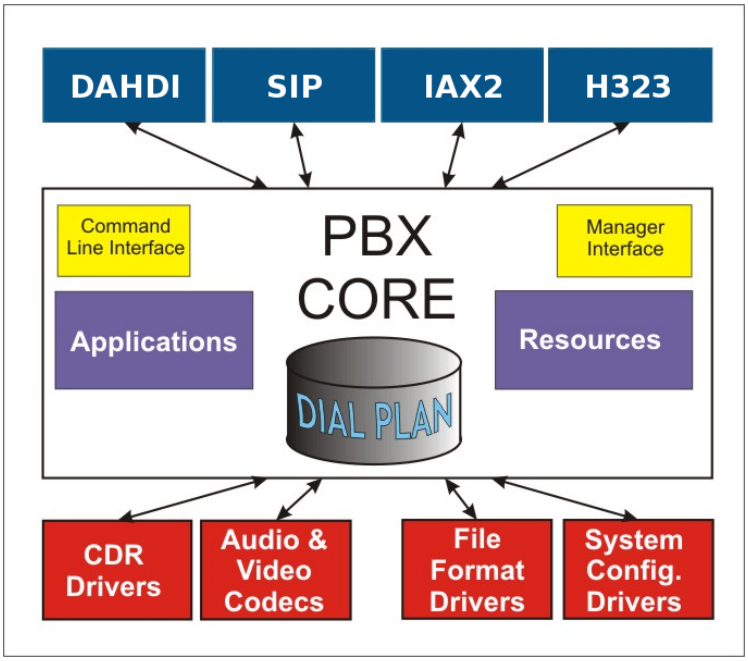


Figura 2.1: Arquitetura do Asterisk [7]

Core

O *core* é o componente central do Asterisk. Este gere a inicialização do sistema, faz a leitura de configurações, carrega módulos e constrói o *Dialplan*. O *core* fornece a base essencial para o bom funcionamento do Asterisk.

Módulos

Os módulos são peças individuais de funcionalidade no Asterisk e para além das aplicações e recursos, estão também retratados a vermelho na Figura 2.1. Podem ser *drivers* de canal, aplicações ou módulos específicos, relacionados a *codecs* ou formatos de áudio e são carregados dinamicamente para adicionar funcionalidades ao sistema. Uma instalação padrão do Asterisk engloba mais de cento e cinquenta módulos distintos [7].

Drivers de Canal

Os *drivers* de canal são módulos responsáveis por lidar com diferentes tecnologias de comunicação, como SIP e H323 e estão representados no topo da Figura 2.1. Estes são responsáveis por fornecer os canais que representam as conexões de comunicação entre dispositivos ou serviços.

Canais

Os canais são instâncias individuais que representam uma ligação de comunicação no Asterisk. São criados pelos *drivers* de canal e podem ser utilizados para gerir chamadas e interagir com aplicações.

Todos estes componentes trabalham em conjunto para criar uma plataforma versátil que pode ser adaptada para atender às necessidades específicas de comunicação de diferentes ambientes e cenários.

2.1.3 *Dialplan* do Asterisk

O *Dialplan* do Asterisk consiste num conjunto crucial de regras e instruções que determinam o fluxo de manipulação de chamadas. Nele, são definidos os procedimentos para receber, processar e encaminhar chamadas, incluindo as ações a serem executadas em cada etapa do processo. A configuração do mesmo é feita através de ficheiros que contêm instruções numa linguagem específica do Asterisk [8].

Tal como referido anteriormente, o *Dialplan* destaca-se como elemento central no funcionamento do serviço Asterisk, desempenhando um papel fundamental no encaminhamento eficiente de chamadas. De seguida, serão abordadas algumas das funcionalidades mais significativas presentes neste serviço.

Contextos, Extensões e Prioridades

No Asterisk, contextos, extensões e prioridades são conceitos fundamentais relacionados à configuração e execução de chamadas. Podem ser classificados da seguinte forma:

- **Contextos:** Contextos são secções isoladas do ficheiro de configuração do Asterisk "`extensions.conf`", também denominado de *Dialplan*. Cada contexto define um conjunto de regras para um determinado encaminhamento de chamadas. É uma forma de organizar e agrupar as configurações relacionadas a um conjunto específico de extensões telefónicas ou serviços [9].

- **Extensões:** Uma extensão representa um conjunto nomeado de ações. Uma extensão pode corresponder a um telefone, a um serviço interno ou a uma série de comandos específicos.

- **Prioridades:** Cada extensão pode ter várias prioridades associadas. As prioridades indicam a ordem pela qual o Asterisk deve executar as ações relacionadas a uma extensão e são usadas para definir o seu fluxo de execução. Por exemplo, se uma chamada chegar a uma determina extensão, o Asterisk seguirá as prioridades de execução para determinar a ordem das ações que devem ser desempenhadas.

Em suma, estes conceitos oferecem flexibilidade e controlo na configuração e operação de um sistema baseado em Asterisk.

Aplicações do *Dialplan*

Na versão mais recente do Asterisk (versão 21), foram incorporadas aproximadamente 221 aplicações ao *Dialplan* [10]. Estas aplicações capacitam os utilizadores a realizar diversas ações durante o processamento de uma chamada, e variam desde funções básicas como chamadas para outros dispositivos, até funcionalidades mais avançadas como manipulação de chamadas, interações com bases de dados ou serviços externos. As aplicações podem ser *blocker*, o que significa que interrompem o fluxo do *Dialplan* até que a ação específica seja concluída. Este tipo de aplicação é crucial para operações que exigem a conclusão de uma tarefa antes de prosseguir para a próxima etapa no *Dialplan*. Por outro lado, existem aplicações que não o são, permitindo que o *Dialplan* continue a ser executado em paralelo enquanto a aplicação realiza as suas tarefas. Esta distinção é fundamental para a elaboração de *Dialplans* eficientes e adequados às necessidades específicas de cada implementação.

Dentro do vasto conjunto de aplicações disponíveis no Asterisk, destaca-se a aplicação **Stasis** [11]. Quando integrada no *Dialplan*, a `Stasis` assume um papel significativo ao proporcionar aos programadores a capacidade de desenvolver soluções dinâmicas e interativas.

A `Stasis` permite uma interação avançada com chamadas em curso, sendo a sua utilização frequentemente observada na implementação de fluxos personalizados de chamadas, sistemas de atendimento automático e na integração do Asterisk com outras plataformas e serviços através de interfaces RESTful.

A `Stasis` possibilita não só a criação de aplicações de comunicação mais robustas, mas oferece também uma abordagem flexível para personalizar e adaptar as funcionalidades do Asterisk de acordo com necessidades específicas.

2.1.4 APIs do Asterisk

As APIs do Asterisk desempenham um papel fundamental no panorama desta plataforma de comunicação de código aberto. Estas interfaces oferecem meios de interação e controlo, permitindo aos desenvolvedores personalizar e integrar soluções telefónicas de maneira flexível. Em conjunto, são essenciais devido ao seu papel na criação de soluções de comunicação adaptáveis e personalizadas. De seguida, proceder-se-á à análise detalhada de cada uma destas interfaces.

AGI

A *Asterisk Gateway Interface* (AGI) assume um papel essencial ao estabelecer conexões entre o *Dialplan* do *Asterisk* e um programa externo. Ao utilizar a AGI, é

possível que este manipule de forma eficaz um canal no *Dialplan*, proporcionando assim uma integração dinâmica entre o sistema e aplicações externas.

Em termos gerais, esta interface é síncrona - as ações realizadas num canal a partir de um bloco AGI não são concluídas até que a ação seja finalizada. Esta característica acrescenta um nível de coesão entre o Asterisk e as aplicações externas, garantindo que as manipulações no canal sejam executadas de forma eficaz e sem interrupções.

Utilizando a AGI, a execução remota do *Dialplan* pode ser ativada, o que permite aos desenvolvedores integrar o Asterisk com *Hypertext Preprocessor* (PHP), Python, Java e outras linguagens de programação [12].

AMI

A *Asterisk Manager Interface* (AMI) é uma componente essencial no ecossistema do Asterisk. Trata-se de uma API que possibilita a interação remota e controlo do Asterisk através de comandos e eventos específicos.

Esta interface possibilita a comunicação bidirecional entre aplicações externas e o Asterisk, permitindo assim uma supervisão em tempo real de eventos e execução de ações específicas no sistema. A AMI opera utilizando o protocolo TCP/IP [13].

Ao contrário da AGI, a AMI adota uma abordagem assíncrona e orientada a eventos, não se concentrando em oferecer meios de controlar a execução de canais. O seu propósito é fornecer informações sobre o estado dos mesmos e controlar onde os canais estão a ser executados [12].

A sua configuração é realizada no ficheiro "`manager.conf`", onde são definidos utilizadores, palavras-passe e as permissões necessárias de leitura e escrita para estabelecer uma ligação remota segura e gerir o sistema de forma controlada.

ARI

A *Asterisk RESTful Interface* (ARI) é uma API que possibilita aos utilizadores desenvolver novas aplicações para o *Dialplan* usando qualquer linguagem de programação compatível com *Hypertext Transfer Protocol* (HTTP) e *WebSockets*. Uma vez que as aplicações são tradicionalmente escritas em C, o objetivo da ARI é ampliar esse acesso, proporcionando uma abordagem mais flexível e acessível para o desenvolvimento de aplicações, permitindo aos desenvolvedores usar uma variedade de linguagens de programação para interagir com o Asterisk de maneira mais eficiente. Os desenvolvedores podem aproveitar funcionalidades como reprodução de toque de anúncios, gravação de áudio, deteção de *Dual-Tone Multi-Frequency*s (DTMFs) e criação e movimentação de canais para dentro e fora de *bridges* para criar as suas aplicações [14].

Além disso, a ARI oferece eventos para notificar quando ocorrem ações, os quais podem ser recebidos passivamente para monitorar aspetos de canais e *bridges* no Asterisk sem a necessidade de interagir diretamente com esses elementos. Essa funcionalidade viabiliza a criação de um painel de atividade em tempo real dentro de um sistema, exibindo eventos como chamadas iniciadas, conexões entre canais ou encerramentos de chamadas.

A ARI é composta por três componentes interconectados: uma interface RESTful para controlo de recursos, um *WebSocket* que transmite eventos *JavaScript Object Notation* (JSON) sobre os recursos do Asterisk ao cliente e a `Stasis` que transfere o controlo de um canal do Asterisk para o cliente. Esses componentes funcionam em harmonia, proporcionando aos desenvolvedores as ferramentas para manipular os recursos fundamentais do Asterisk e construir aplicações de comunicação personalizadas [12].

Embora a AMI seja eficaz no controlo de chamadas e a AGI seja útil para permitir que um processo remoto execute aplicativos do *Dialplan*, nenhuma dessas APIs foi projetada com o propósito de permitir que desenvolvedores construam as suas próprias aplicações personalizadas de comunicação. A ARI foi especificamente desenvolvida para abordar essas preocupações. A Figura 2.2 demonstra de que forma as três APIs estão interligadas.

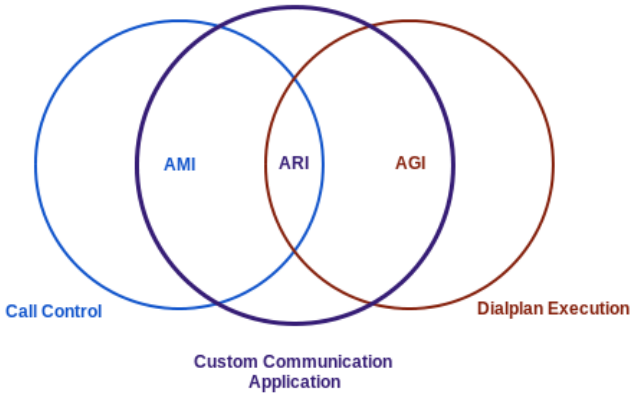


Figura 2.2: Interligação das APIs [12]

Através da Figura 2.2, é possível identificar uma interconexão entre as três APIs - ARI, AMI e AGI - estabelecendo assim uma complexa rede de comunicação e interação do Asterisk. A posição central da ARI na figura realça a incorporação de funcionalidades tanto da AMI como da AGI.

Esta representação gráfica demonstra uma abordagem integrada, onde as APIs comunicam e colaboram entre si para fornecer funcionalidades avançadas a um determinado sistema, tais como a gestão e processamento de chamadas e eventos.

Esta interseção demonstra a existência de áreas com funcionalidades compartilhadas, ilustrando como as APIs podem interagir para atender às várias necessidades de comunicação.

AEAP

A *Asterisk External Application Protocol* (AEAP) é uma API *framework* projetada para estabelecer a ligação e a comunicação com aplicações externas ao Asterisk, além de possibilitar a interação com os seus subsistemas internos [15]. Uma das principais vantagens da AEAP é sua capacidade de gerir conexões, enviar e receber mensagens, assim como realizar a serialização e desserialização de dados. Este conjunto de funcionalidades proporciona um meio eficiente para ativar eventos entre o Asterisk e aplicações externas, sendo particularmente relevante em cenários que requerem comunicação bidirecional.

Embora a AEAP represente uma adição recente ao conjunto de ferramentas do Asterisk, é de salientar a sua constante evolução. A sua estrutura foi concebida para facilitar a integração de novas funcionalidades e melhorias de forma eficiente e direta. Essa flexibilidade possibilita uma adaptação contínua às exigências do mercado e às necessidades dos utilizadores, garantindo que o Asterisk se mantém como uma solução sólida e atualizada.

Estas são algumas das APIs oferecidas pelo Asterisk, cada uma projetada para propósitos específicos e adaptada para diferentes casos de uso. A escolha da API mais apropriada irá depender das necessidades específicas do projeto e dos recursos associados ao mesmo.

2.2 WMS

O WMS é um servidor multimédia desenvolvido pela Altice Labs capaz de assumir o papel de *IP Media Server* e *Media Resource Function* (MRF). Destaca-se pelo processamento avançado de diversos tipos de media, incluindo a reprodução e gravação de anúncios de áudio, adaptação (transcodificação, taxa de bits variável, redimensionamento) e manipulação de conteúdos áudio em tempo real. Além disso, suporta a criação e publicação de serviços de telefonia e de *streaming* de áudio, o que permite uma rápida implementação de serviços multimédia em tempo-real.

2.2.1 Cenários de Utilização

O WMS oferece diversas possibilidades de utilização, adaptando-se à funcionalidade requerida em diferentes cenários. Abaixo destacam-se alguns dos variados contextos nos quais o WMS pode ser implementado.

IP Media Server

Atuando como um *IP Media Server*, o WMS pode ser integrado em diversas topologias de rede *Internet Protocol* (IP). Conforme ilustrado na Figura 2.3, ele pode ser conectado a *softswitches*, servidores e *proxies* SIP, ou qualquer outro equipamento SIP, possibilitando a interação por meio de DTMF ou reconhecimento de fala.

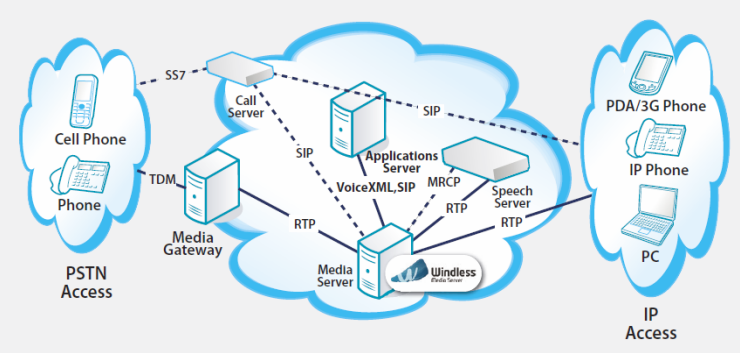


Figura 2.3: WMS como *IP Media Server* [16]

Media Resource Function

O WMS pode também ser apresentado como um MRF, ligado a qualquer *Serving-Call Session Control Function* (S-CSCF) ou *Aplication Server SIP* (SIP AS) em arquiteturas *IP Multimédia Subsystem* (IMS), tal como demonstrado na Figura 2.4.

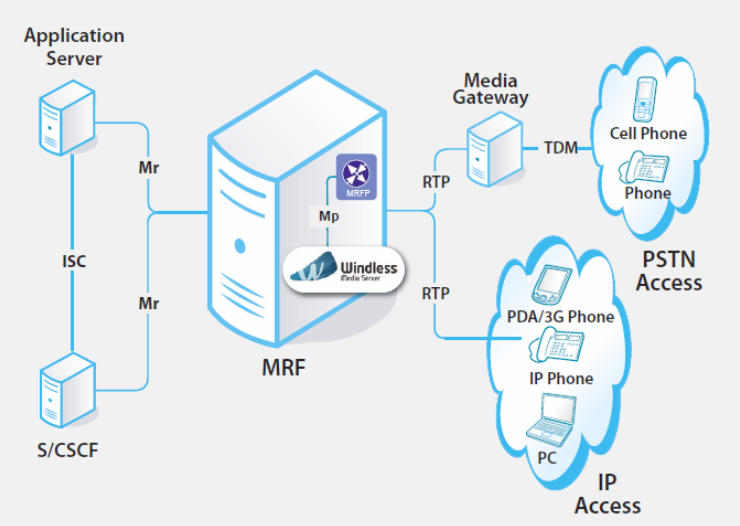


Figura 2.4: WMS como *Media Resource Function* [16]

2.2.2 Arquitetura Lógica

Projetada para evoluir de acordo com as exigências do mercado das telecomunicações, a estrutura modular do WMS tem como objetivo facilitar a integração de recursos de diferentes fornecedores de tecnologia. Na Figura 2.5 é possível observar a arquitetura lógica do WMS.

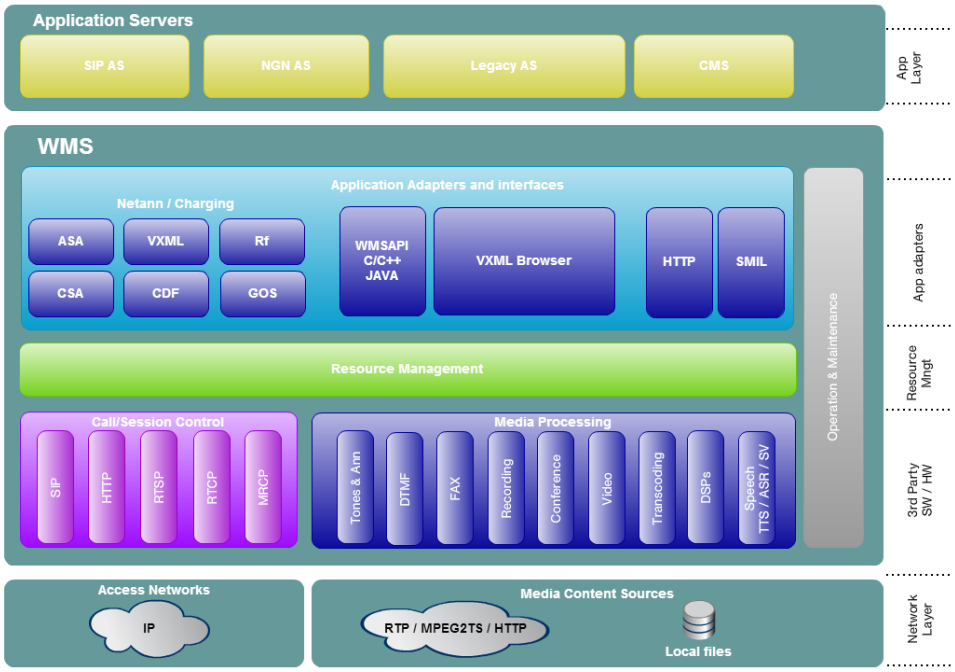


Figura 2.5: Arquitura lógica do WMS [17]

A arquitetura lógica está dividida nas seguintes camadas:

- **Camada de Rede:** A camada de rede tem como principal objetivo possibilitar a comunicação entre dispositivos através da transmissão de pacotes de dados. Esta camada é responsável por encaminhar pacotes de dados de um dispositivo de origem para um dispositivo de destino.
- **Camada do WMS:** A camada WMS desempenha um papel crucial na integração das funcionalidades de controlo de chamadas e processamento de media. No que diz respeito ao processamento de media, o WMS disponibiliza recursos avançados como ASR, TTS e criação de conferências de áudio. Para tal, o WMS conta com um módulo dedicado à gestão destes recursos específicos. Em termos de *adapters* e interfaces, o WMS oferece a API WMS, que facilita a criação de serviços em Java. Adicionalmente, disponibiliza mecanismos robustos para a criação de serviços em *VoiceXML* (VXML).

- **Camada da Aplicação:** A camada da aplicação tem como principal objetivo fornecer interfaces e serviços de comunicação.

2.2.3 Módulo CMAN

O módulo *Channel Management* (CMAN) representa o *core* telefónico do WMS, cuja principal função é assegurar a gestão eficiente de chamadas SIP. É o *core* do *IP Media Server*. Na Figura 2.6 observa-se a arquitetura funcional do CMAN.

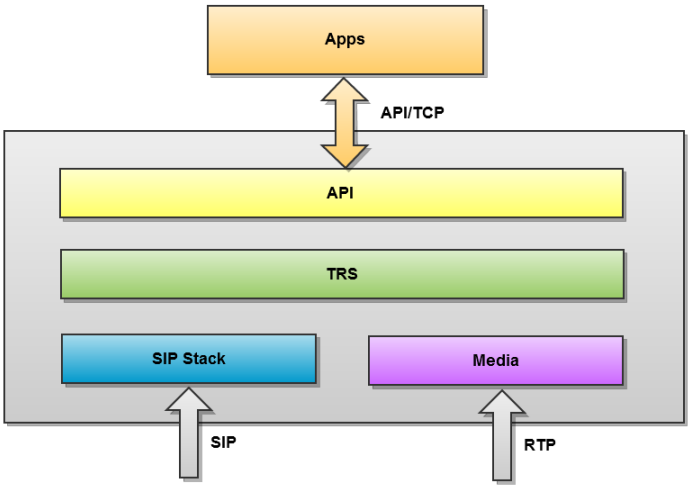


Figura 2.6: Arquitetura funcional do CMAN [18]

- **Aplicações (*Apps*):** No topo da arquitetura, as aplicações representam os serviços e funcionalidades oferecidos pelo CMAN. Estas aplicações podem incluir serviços de telefonia, conferências, mensagens de voz, entre outros.
- **API:** A camada relativa à API atua como um intermediário entre as aplicações e os componentes subjacentes do CMAN. A comunicação entre as aplicações e a API ocorre via protocolos API/TCP, garantindo a interoperabilidade e a troca eficiente de informações.
- ***Telephony Resource Server* (TRS):** Logo abaixo da API, encontra-se a camada TRS, que é responsável pela gestão dos recursos de telecomunicações. Esta camada facilita a alocação e a gestão de recursos como canais de voz, *codecs*, e outras funcionalidades necessárias para o processamento das chamadas.
- **SIP Stack e Media:** Na base da arquitetura, a pilha SIP é crucial para a sinalização das chamadas. Esta camada cuida do estabelecimento, manutenção e encerramento das sessões de comunicação, utilizando o protocolo SIP.

Paralelamente a esta, a camada de Media lida com o transporte dos dados em tempo real. Utilizando o protocolo *Real-Time Transport Protocol* (RTP), esta camada garante a entrega de media, como áudio e vídeo, com baixa latência.

A Figura 2.7 detalha a arquitetura lógica do CMAN, elucidando os diversos módulos funcionais e as suas interações.

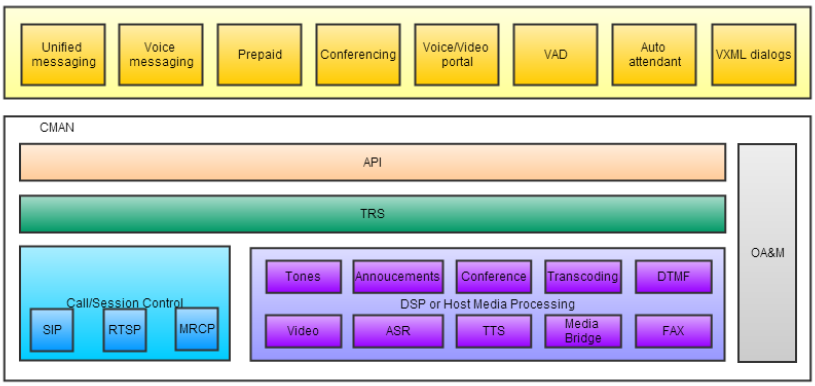


Figura 2.7: Arquitetura lógica do CMAN [18]

- **Serviços CMAN:** No topo, estão listados vários serviços suportados pelo CMAN, tais como *unified messaging*, mensagens de voz, pré-pagos e conferências. Estes serviços fornecem uma ampla gama de funcionalidades que são essenciais para a operação eficiente do sistema.
- **API:** A API, semelhante à Figura 2.6, serve de interface entre os serviços do CMAN e os componentes subjacentes, facilitando a comunicação e a integração.
- **TRS:** A camada TRS continua a desempenhar o papel de gestão dos recursos de telecomunicações, crucial para o funcionamento eficiente dos serviços.
- **Componentes de Media e Sessão:** Os componentes de media e sessão incluem módulos como anúncios, transcodificação, ASR e TTS, DTMF e *media bridges*. Além disso, a camada de controlo de chamadas e sessões abrange diversos protocolos essenciais para a gestão das comunicações e sessões multimédia.
- **OA&M (Operação, Administração e Gestão):** Esta componente assegura a operação e manutenção do sistema, garantindo que todos os módulos funcionem corretamente e que as falhas sejam rapidamente resolvidas. Este módulo é vital para a estabilidade do sistema, assegurando uma performance contínua e sem interrupções.

Estas figuras demonstram a robustez e a versatilidade do CMAN, evidenciando como a sua arquitetura bem definida é essencial para suportar e integrar uma ampla gama de serviços de telecomunicações e media no WMS.

2.3 Protocolos de Comunicação Multimédia

Nas comunicações modernas, o *Session Initiation Protocol* (SIP), *PJ Session Initiation Protocol* (PJSIP), *Session Description Protocol* (SDP), *Real-Time Transport Protocol* (RTP) e *Media Resource Control Protocol* (MRCP) desempenham papéis cruciais na simplificação de serviços como chamadas de voz pela Internet (VoIP), videoconferências e transmissão de media em tempo real. Enquanto o PJSIP, que incorpora o SIP, lida com a sinalização, o *Session Description Protocol* (SDP) trata da descrição da sessão. O RTP, por sua vez, é responsável pelo transporte eficiente dos dados em tempo real. Já o MRCP é um protocolo utilizado para controlar recursos de media, como servidores de reconhecimento de fala ou síntese de voz, em sistemas de comunicação multimédia.

Para uma compreensão mais abrangente das suas funcionalidades e relevância, cada protocolo será detalhadamente explorado nesta seção.

2.3.1 SIP

Segundo o *Request for Comments* (RFC) 3261 do *Internet Engineering Task Force* (IETF) [19], o *Session Initiation Protocol* (SIP) é um protocolo da camada de aplicação que permite criar, modificar e terminar sessões de comunicação multimédia. Criado em 1999 por Hennin Schulzrinne, Mark Handley, Eve Schooler e Jonathan Rosenberg, é amplamente utilizado como o protocolo de sinalização padrão para tecnologias VoIP devido à sua flexibilidade, capacidade de integração e compatibilidade com diversos dispositivos.

Uma das principais vantagens do SIP é a sua capacidade de ser adotado como um protocolo universal e comum, onde qualquer fabricante tem a liberdade de implementar o protocolo nos seus dispositivos ou sistemas sem restrições. Ao adotar esta abordagem, o protocolo SIP exerce um impacto significativo na promoção da interoperabilidade entre uma ampla gama de equipamentos e serviços.

Arquitetura SIP

A arquitetura SIP é formada por quatro principais componentes que trabalham em conjunto para garantir a eficácia e fiabilidade de comunicações sobre IP. Cada um desempenha funções únicas, desde a origem e encaminhamento de chamadas até a manutenção do estado da sessão e a garantia de conectividade adequada. Estes são:

- ***User Agent* (UA):** O UA é um componente central no SIP que engloba os *User Agent Client*s (UACs) e o *User Agent Server* (UAS) [19]. O UAC é encarregado de iniciar solicitações, como registar, convidar, encerrar ou cancelar. O UAS, por sua vez, é uma aplicação de servidor que entra em contacto com o utilizador quando recebe solicitações SIP, respondendo em nome do utilizador. Com base na resposta recebida, a solicitação pode ser aceite, rejeitada ou redirecionada.

- **Servidor de Registo:** O servidor de registo é responsável por rastrear e armazenar informações sobre a localização e prioridades dos utilizadores na rede. Quando um UA deseja registar na rede, é enviado um pedido de registo ao servidor de registo, informando a sua localização atual.

- **Servidor *Proxy*:** O servidor *proxy* atua como intermediário entre os UA, ajudando a encaminhar solicitações e respostas SIP pela rede. Os pedidos podem ser realizados diretamente ou através de um outro servidor que esteja mais próximo do recetor.

- **Servidor de Redirecionamento:** O servidor de redirecionamento auxilia na indicação do caminho correto para uma solicitação SIP. Quando um servidor *proxy* ou UAS decide que a solicitação deve ser redirecionada para outro local, ele retorna uma resposta de redirecionamento, indicando o novo endereço ou servidor para onde a solicitação deve ser encaminhada.

A Figura 2.8 demonstra uma possível organização desta arquitetura.

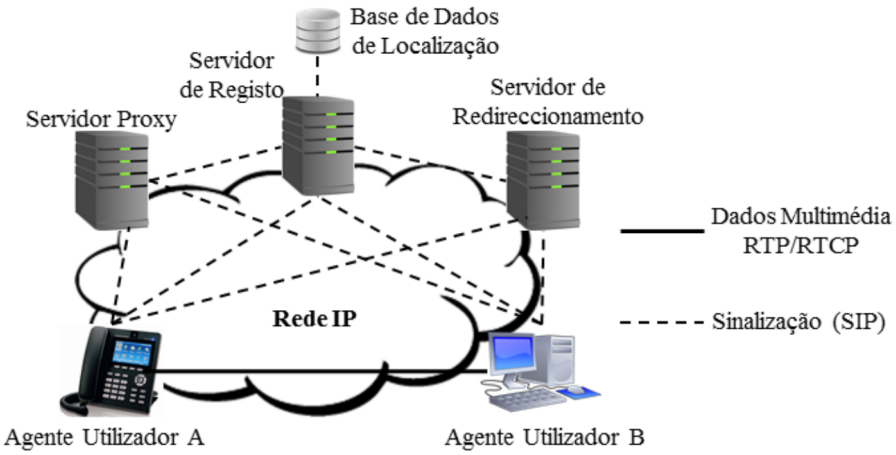


Figura 2.8: Arquitetura geral do SIP [20]

Estes elementos em conjunto formam a base para uma arquitetura SIP robusta e escalável, capaz de suportar uma variedade de aplicativos de comunicação modernos.

Métodos Fundamentais

Os métodos fundamentais do SIP descrevem as ações que podem ser realizadas numa sessão SIP [21]. Estes oferecem vantagens significativas para comunicações sobre IP, permitindo o estabelecimento, modificação e encerramento eficientes de sessões de comunicação. Estes métodos são:

- **`INVITE`:** Usado para iniciar uma sessão de comunicação entre UA, podendo ter informações de media do chamador no corpo da mensagem. Uma sessão é considerada estabelecida se um `INVITE` receber uma resposta de sucesso ou se um `ACK` for enviado.
- **`BYE`:** O UA utiliza o `BYE` para comunicar ao servidor o seu desejo de finalizar a chamada. Esse pedido, semelhante ao `INVITE`, pode ser originado tanto pelo chamador quanto pelo chamado.
- **`REGISTER`:** Permite registar um UA. Este pedido é enviado de um UA para um servidor registador.
- **`CANCEL`:** Usado para terminar uma sessão que não está estabelecida. Os UA utilizam este pedido para cancelar uma tentativa de chamada pendente iniciada anteriormente.
- **`ACK`:** Confirma que o cliente recebeu uma resposta final a um `INVITE`.
- **`OPTIONS`:** É utilizado para consultar as capacidades de um UA ou servidor *proxy*, assim como determinar a sua disponibilidade atual. A resposta a essa solicitação detalha as capacidades do UA ou do servidor. Além disso, é uma exigência que este método seja suportado por servidores *proxy*, servidores de redirecionamento, UA, registadores e clientes SIP.

Fluxo Básico de Chamadas

Um fluxo básico de chamadas SIP pode ser dividido em três etapas distintas: estabelecimento da sessão, transferência de media e encerramento da sessão. A imagem 2.9 demonstra uma possível interação utilizando o protocolo SIP.

Durante o estabelecimento da sessão, o UA A inicia o processo enviando uma mensagem de `INVITE` para o UA B, que responde com um `180 Ringing` e um `200 OK`, indicando que o pedido foi recebido e está a ser processado. Este fica apenas à espera de um `ACK` por parte do outro UA de forma a finalizar o estabelecimento da sessão.

Durante a transferência de media, os UAs A e B transferem dados utilizando os protocolos RTP e *Real-Time Transport Control Protocol* (RTCP) para garantir uma comunicação eficiente e em tempo real.

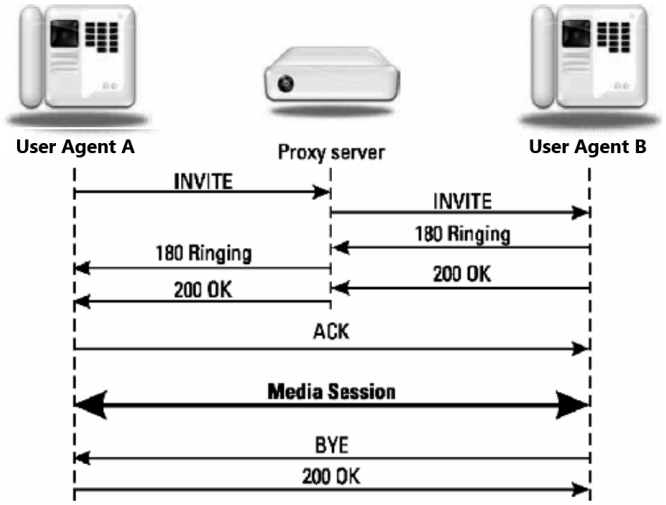


Figura 2.9: Cenário de sinalização de uma chamada [22]

Após a transferência dos dados, o UA B inicia o encerramento da sessão enviando uma mensagem `BYE` para o UA A, indicando a sua intenção de terminar a comunicação. Em resposta, o UA A envia uma mensagem `200 OK`, confirmando o encerramento da sessão por sua parte.

Em conjunto com os protocolos SDP e RTP, o SIP fornece a sinalização necessária para estabelecer, modificar e encerrar sessões de comunicação em tempo real.

2.3.2 SDP

O *Session Description Protocol* (SDP) é frequentemente utilizado com o protocolo SIP para descrever parâmetros de uma determinada sessão de comunicação num formato compreendido por todos os participantes. De acordo com o RFC 4566 [23], o SDP oferece uma padronização para a transmissão de detalhes sobre media, endereços de transporte e dados essenciais para a descrição de sessões, sendo exclusivamente um formato para descrever sessões, desvinculado-se de qualquer protocolo de transporte específico. O SDP é destinado a ser de uso geral para que possa ser usado numa ampla variedade de ambientes e aplicações de rede.

Alguns dos principais exemplos de utilização deste protocolo são:

- **Inicialização de Sessão:** As mensagens SIP transportam descrições detalhadas de sessões, permitindo que os participantes concordem com um conjunto de tipos de media compatíveis. Essas descrições são frequentemente formatadas usando o SDP que oferece um meio padronizado para especificar parâmetros como *codecs* de áudio, endereços IP e outras informações relevantes.

- ***Streaming* de media:** Os servidores de streaming de media utilizam o SDP para descrever as características da media a ser transmitidas como formato, taxa de bits e o *Uniform Resource Identifier* (URI) de transmissão.
- **Anúncios de Sessão *Multicast*:** Os anúncios de sessão *multicast* são mensagens projetadas para divulgar a disponibilidade e os detalhes de uma sessão de comunicação *multicast* numa rede específica. O SDP é responsável por fornecer o formato recomendado de descrição da sessão para tais anúncios.

Especificações do SDP

Ao utilizar o protocolo SDP, é necessário enviar uma mensagem com informações essenciais sobre a sessão a estabelecer. Uma descrição de sessão em SDP consiste num número determinado de linhas na forma `<tipo> = <valor>`. Um exemplo detalhado dessa estrutura é fornecido na Listagem 2.1, onde são apresentadas todas as possíveis linhas de descrição de sessão em SDP.

É possível dividir essa mensagem em três categorias distintas: **descrição da sessão**, que abrange características gerais da sessão, como versão do protocolo, nome e criador, podendo ainda incluir informações de contacto, largura de banda disponível, fuso horário e criptografia; a **descrição de tempo**, que incide sobre os parâmetros relacionados com a duração da sessão, como a hora de início e término, assim como a repetição de padrões temporais, se aplicável; e a **descrição de media**, onde são detalhados os parâmetros de fluxo de media presente na sessão, como o *codec* utilizado, endereço de transporte e porta para a transmissão, entre outros detalhes específicos da media. As linhas opcionais de descrição estão identificados através de um "`=*`" a separar o "tipo" e "valor".

```
Descrição da sessão:
        v = (versão do protocolo)
        o = (criador e identificador de sessão)
        s = (nome da sessão)
        i =* (sessão de informação)
        u =* (URI de descrição)
        e =* (endereço de email)
        p =* (número de telefone)
        c =* (informações de conexão)
        b =* (informação de largura de banda)
        z =* (ajustes de fuso horário)
        k =* (chave de criptografia)
        a =* (atributo de sessão)

Descrição do tempo:
        t = (tempo que a sessão está ativa)
        r =* (repetir)

Descrição da media:
        m = (nome da media e endereço de transporte)
        i =* (título da media ou campo de informação)
```

```
c =* (informações da conexão)
b =* (informação de largura de banda)
k =* (chave de criptografia)
a =* (atributos da media)
```

Listagem 2.1: Parâmetros de descrição [23]

Embora nem todas as linhas sejam obrigatórias, é crucial que cada descrição de sessão seja organizada conforme a ordem indicada. Essa estrutura normalizada facilita a deteção de erros e promove uma leitura mais fluente e coesa da mensagem.

2.3.3 RTP

O *Real-Time Transport Protocol* (RTP), definido pela RFC 3550 [24], é um protocolo de transporte em tempo real, utilizado para transmitir dados multimédia, como áudio e vídeo, através de redes IP. É um protocolo da camada de transporte que funciona sobre o *User Datagram Protocol* (UDP) [25]. Apesar de não garantir a entrega de dados, o RTP fornece mecanismos para sincronização e recuperação de pacotes perdidos, incorporando informações como números de sequência de pacotes e *timestamps* no seu cabeçalho. Com isso, é possível que uma aplicação recetora faça o armazenamento em *buffer* e a sequenciação dos pacotes na ordem correta [26]. Na Tabela 2.1 está representado o formato do cabeçalho de um pacote RTP.

Bit Offset	**0-1**	**2**	**3**	**4-7**	**8**	**9-15**	**16-31**
0	Version	Padding	Ext.	CSRC Count	Marker	Payload Type	Sequence Number
32	Timestamp						
64	SSRC Identifier						
96	CSRC Identifier						
96+32*CC	Payload						

Tabela 2.1: Cabeçalho do protocolo RTP [26]

Neste cabeçalho é possível identificar a versão do RTP, a possibilidade de acrescentar extensões ao cabeçalho original, o número de identificadores *Contributing Source* (CSRC), entre outros. São de salientar os campos destinados ao *Synchronization Source* (SSRC) *Identifier* e ao *Contributing Source* (CSRC) *Identifier*, que identificam a origem da sincronização e a fonte das contribuições individuais que compõem a carga útil do fluxo de dados para o pacote, respetivamente.

RTCP

Em conjunto com o RTP, o *Real-Time Transport Control Protocol* (RTCP) fornece um *feedback* da qualidade dos serviços oferecidos pelo RTP. Enquanto o RTP realiza a entrega dos dados, o RTCP envia pacotes de controlo aos participantes. Em cenários onde a qualidade da transmissão é crucial, é recomendado utilizar ambos os protocolos em conjunto. O RTCP fornece informações valiosas sobre a qualidade da transmissão, como estatísticas de tráfego, relatórios de receção e *feedback* sobre a experiência do utilizador. No entanto, em ambientes onde a comunicação bidirecional não é necessária, pode ser viável utilizar apenas o RTP, como por exemplo em transmissões de áudio ou vídeo em *multicast*, onde o *feedback* de qualidade ou o controlo de fluxo não são essenciais.

O RTCP realiza quatro principais funções [24]: (1) fornecer *feedback* sobre a qualidade da distribuição de dados; (2) movimentar um identificador de transporte para uma fonte RTP - *Canonical Name* (CNAME); (3) controlar a taxa de envio de dados, evitando congestionamento na rede; (4) transmitir informações mínimas de controlo de sessão, como identificação de participantes.

Assim, é possível haver uma monitorização da qualidade da transmissão, bem como fazer uma gestão eficiente de sessões em tempo real.

Para enviar uma determinada informação de controlo, o RTCP utiliza os seguintes tipos de pacote:

- ***Sender Report* (SR):** Enviado pelo remetente - fornece estatísticas de envio e receção de pacotes RTP.
- ***Receiver Report* (RR):** Enviado pelos recetores - fornece estatísticas de receção de pacotes RTP.
- ***Source Description Items* (SDES):** Contém informações sobre os participantes da sessão (CNAME, endereço IP e tipo de media).
- ***Goodbye* (BYE):** Indica que um participante saiu da sessão.
- ***Application-defined* (APP):** Usado para transportar informações específicas da aplicação.

A consolidação destes pacotes em um único é uma prática aceitável para reduzir o tráfego. No entanto, é crucial notar que um pacote consolidado deve iniciar com um SR ou RR. Além disso, se um pacote contiver um `BYE`, este deve ser obrigatoriamente o último item contido no pacote.

2.3.4 PJSIP

PJ Session Initiation Protocol (PJSIP) é uma biblioteca multimédia de código aberto escrita em C que implementa protocolos de comunicação de áudio e vídeo, tais

como SIP, SDP e RTP. Desenvolvida para ser eficiente e flexível, é amplamente utilizada em aplicações que requerem comunicação em tempo real, como sistemas de VoIP e videoconferência [27]. Esta combina o protocolo SIP com uma *framework* multimédia e funcionalidades *Network Address Translation* (NAT) numa API de alto nível [28].

A arquitetura do PJSIP é modular, permitindo fácil extensibilidade e manutenção. A sua arquitetura está demonstrada na Figura 2.10.

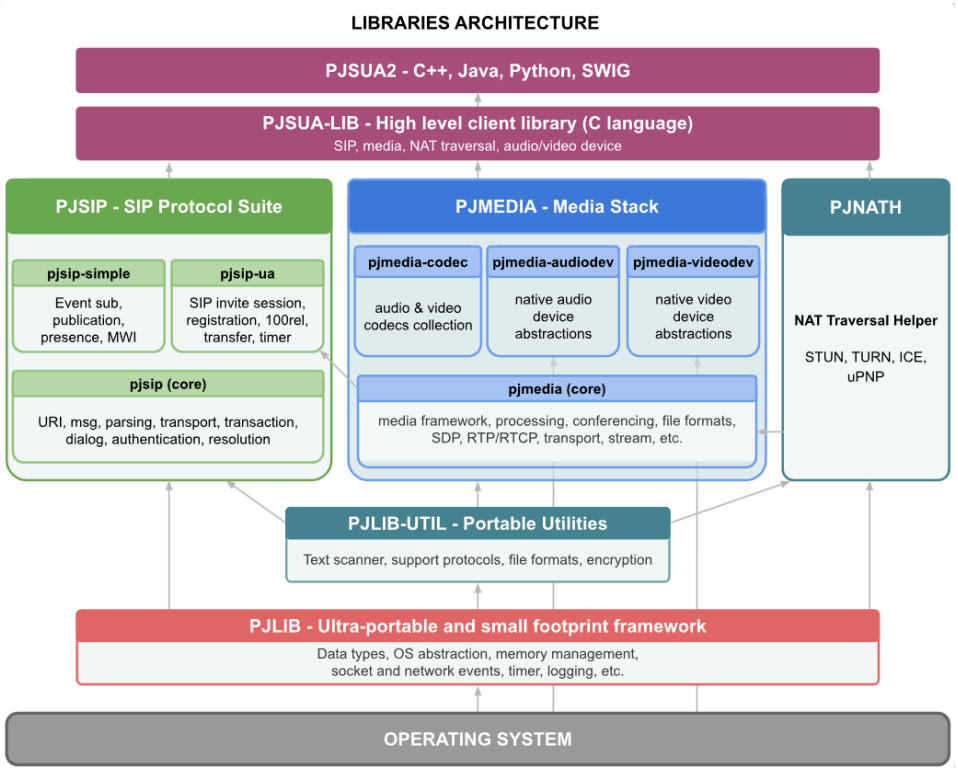


Figura 2.10: Arquitetura do PJSIP [28]

Podemos dividir a arquitetura do PJSIP nos seguintes componentes [28]:

- **PJSUA-LIB:** Interface de alto nível que facilita a construção de aplicações SIP, gerindo contas, chamadas e mensagens instantâneas.
- **PJMEDIA:** Biblioteca multimédia que lida com a codificação/descodificação de áudio e vídeo, transporte RTP/RTCP e processamento de áudio (como cancelamento de eco e ajuste de volume).
- **PJNATH:** Biblioteca para travessia de NAT, implementando protocolos como STUN, TURN e ICE para garantir a sua conectividade.
- **PJLIB:** Biblioteca utilitária que fornece estruturas de dados, manipulação de *strings*, gestão de memória, operações de rede e suporte a programação *multithread*.

A combinação destas bibliotecas permite ao PJSIP fornecer uma solução completa e robusta para aplicações de comunicação em tempo real. Além disso, a sua portabilidade tornam-no uma escolha ideal para desenvolvedores que procuram implementar funcionalidades de comunicação em diferentes plataformas e dispositivos.

2.3.5 MRCP

O *Media Resource Control Protocol* (MRCP), conforme definido pela RFC 4463 [29], é uma tecnologia que proporciona acesso a recursos avançados de processamento de multimédia, como reconhecimento e síntese de fala, através de redes IP. Ao utilizar tecnologias como SIP, HTTP e *Extensible Markup Language* (XML) [30], o MRCP oferece uma interface flexível e *open standard*, simplificando o processo de integração de tecnologias de fala em equipamentos de rede, permitindo que operadoras e fornecedores proporcionem serviços interativos de maneira mais eficiente. Existem atualmente duas versões deste protocolo: MRCPv1 e MRCPv2. O MRCPv2, versão mais recente, introduz melhorias e funcionalidades adicionais em relação ao MRCPv1, tais como a presença de uma comunicação mais eficiente, suporte a uma variedade mais ampla de recursos multimédia, como reconhecimento e síntese de fala avançados, além de uma maior flexibilidade na implementação de serviços interativos de fala em redes IP [31].

Arquitetura do MRCP

O MRCP foi desenvolvido com o propósito de facultar aos clientes a capacidade de invocar e gerir recursos especializados de processamento de media dentro de uma rede. Este baseia-se no protocolo SIP, cujo objetivo é localizar um *Media Resource Server* (MRS) para estabelecer canais de comunicação. Este servidor é responsável por disponibilizar uma variedade de recursos, tais como reconhecimento, síntese, verificação e gravação de voz.

O SIP permite que um cliente MRCP localize um tipo de recurso na rede através de um SIP URI e consulte as capacidades de um MRS. Uma vez encontrado um servidor adequado, o SIP é utilizado para estabelecer duas vias de comunicação: uma para enviar/receber áudio (sessão de media) e outra para controlar o MRS e receber eventos (sessão de controlo). A Figura 2.11 demonstra uma possível representação da arquitetura do protocolo MRCP.

O cliente MRCP é composto tanto por uma pilha SIP quanto por uma pilha MRCP. Quando a camada de aplicação requer um recurso de fala, esta invoca a *Media Resource API*, que, por sua vez, estabelece uma comunicação SIP com o MRS através da pilha SIP. Este processo dá início a uma sessão multimédia que utiliza o protocolo RTP, além de uma sessão de controlo que ocorre sobre o protocolo

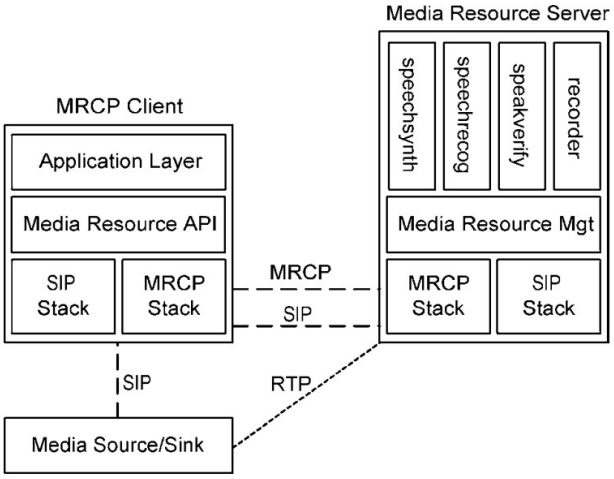


Figura 2.11: Arquitetura do MRCP [30]

MRCP com o MRS. Já a pilha MRCP assume o papel de servidor, respondendo a solicitações do cliente, gerando eventos na direção do mesmo [30].

Neste contexto, podemos apresentar algumas situações para ilustrar as características fundamentais do protocolo MRCP. O primeiro exemplo descreve o fluxo essencial de mensagens destinadas a um sintetizador de fala, enquanto o segundo delineia o fluxo primordial de mensagens para um reconhecedor de fala.

Papel do MRCP num Sintetizador de Fala

A Figura 2.12 ilustra uma troca de mensagens típica do protocolo MRCP, entre um cliente MRCP e um MRS que incorpora recursos de síntese de fala.

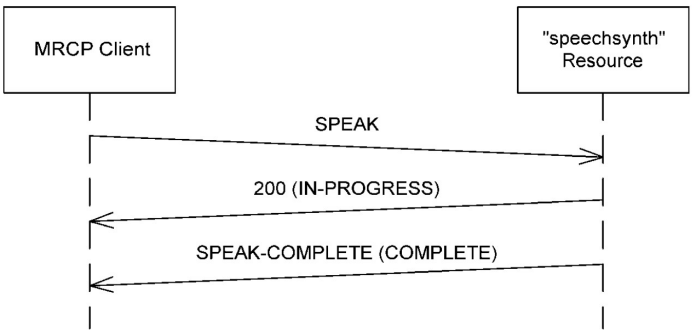


Figura 2.12: MRCP como sintetizador de fala [30]

A transmissão das mensagens é realizada utilizando os protocolos *Transmission Control Protocol* (TCP) ou *Stream Control Transmission Protocol* (SCTP). A fase atual da solicitação é indicada entre parênteses para fornecer um *feedback* claro ao utilizador. No contexto do cliente MRCP, o processo é iniciado com o envio de

uma solicitação SPEAK para o MRS. Uma possível representação desta mensagem é a seguinte [30]:

```
MRCP/2.0 380 SPEAK 14321
Channel-Identifier: 43b9ae17@speechsynth
Content-Type: application/ssml+xml
Content-Length: 253

<?xml version="1.0" encoding="UTF-8"?>
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis">
<emphasis>Good afternoon</emphasis> Anne. <break/>
You have one voice message, two e-mails, and three faxes
waiting for you.
</speak>
```

As mensagens de solicitação do MRCP seguem um formato padrão, dividido em três secções principais: a linha de solicitação, os campos do cabeçalho e, opcionalmente, o corpo da mensagem. A linha de solicitação inicia-se com o nome e a versão do protocolo (MRCP/2.0), seguidos pelo comprimento da mensagem (380) em bytes e pelo nome do método da solicitação (SPEAK). Um ID de solicitação único é incluído para correlacionar respostas e eventos (14321). Após a linha de solicitação, seguem-se os campos do cabeçalho, como o identificador do canal que inclui o sufixo *speechsynth*, e, caso exista um corpo de mensagem, devem ser especificados os cabeçalhos Content-Type e Content-Length. Uma possível resposta a esta mensagem poderá ser a seguinte [30]:

```
MRCP/2.0 119 14321 200 IN-PROGRESS
Channel-Identifier: 43b9ae17@speechsynth
Speech-Marker: timestamp=857206027059
```

O código 200 denota o sucesso do pedido, enquanto este se encontra no estado IN-PROGRESS, indicando que a transmissão de áudio está atualmente em progresso.

Por fim, é gerado o evento SPEAK-COMPLETE pelo MRS e a solicitação transita para o estado COMPLETE, indicando que não serão enviados mais eventos [30]:

```
MRCP/2.0 157 SPEAK-COMPLETE 14321 COMPLETE
Channel-Identifier: 43b9ae17@speechsynth
Speech-Marker: timestamp=861500994355
Completion-Cause: 000 normal
```

Neste exemplo, o valor do Completion-Cause indica que a solicitação SPEAK foi terminada de forma expectável.

Papel do MRCP num Reconhecedor de Fala

A Figura 2.13 ilustra uma troca de mensagens típica do protocolo MRCP, entre um cliente MRCP e um MRS que incorpora recursos de reconhecedor de fala.

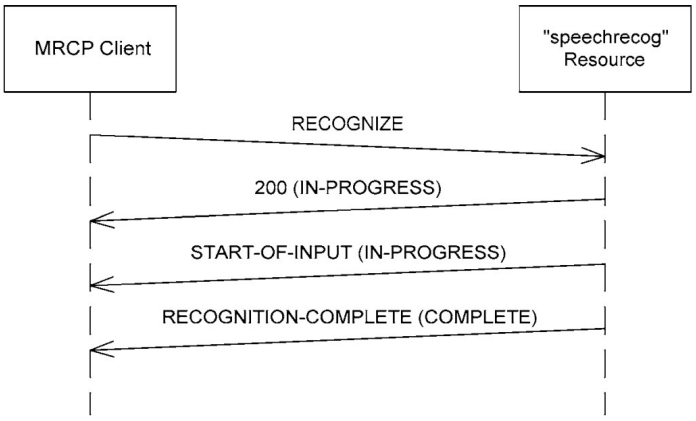


Figura 2.13: MRCP como reconhecedor de fala [30]

O formato de um pedido `RECOGNIZE` é semelhante ao formato da solicitação `SPEAK` anteriormente apresentada. Uma possível representação deste pedido é o seguinte [30]:

```
MRCP/2.0 461 RECOGNIZE 32121
Channel-Identifier: 23af1e13@speechrecog
Content-ID: <grammar1@form-level.store>
Content-Type: application/srgs+xml
Content-Length: 289
<?xml version="1.0" encoding="UTF-8"?>
<grammar version="1.0" xmlns="http://www.w3.org/2001/06/grammar"
xml:lang="en-GB">
<rule id="yesno">
<one-of>
<item>yes</item>
<item>no</item>
</one-of>
</rule>
</grammar>
```

Ao contrário de uma solicitação `SPEAK`, o identificador de canal inclui o sufixo *speechrecog*, indicando que se trata de um recurso específico de reconhecimento de fala. No corpo do pedido, uma gramática de fala é incorporada utilizando o formato XML para delinear palavras e frases passíveis de reconhecimento. Neste exemplo, o reconhecedor é limitado a duas alternativas: *"yes"* ou *"no"*. Uma potencial resposta do MRS está ilustrada abaixo [30]:

```
MRCP/2.0 79 32121 200 IN-PROGRESS
Channel-Identifier: 23af1e13@speechrecog
```

Esta resposta assemelha-se ao exemplo fornecido para o pedido `SPEAK`: é uma resposta bem-sucedida (código `200`), indicando ao cliente que o pedido está em curso e que o reconhecedor de fala está a recolher áudio no momento. Assim que a fala é detetada, é desencadeado o evento `START-OF-INPUT` [30]:

```
MRCP/2.0 111 START-OF-INPUT 32121 IN-PROGRESS
Channel-Identifier: 23af1e13@speechrecog
Input-Type: speech
```

Este evento é normalmente utilizado pelo cliente para interromper o sintetizador de voz. Posteriormente, quando o reconhecedor de fala terminar o processo de reconhecimento, é emitido o evento `RECOGNITION-COMPLETE` que inclui os resultados de reconhecimento no corpo da mensagem associada [30]:

```
MRCP/2.0 472 RECOGNITION-COMPLETE 32121 COMPLETE
Channel-Identifier: 23af1e13@speechrecog
Completion-Cause: 000 success
Content-Type: application/nlsml+xml
Content-Length: 289
<?xml version="1.0" encoding="UTF-8"?>
<result grammar="session:grammar1@form-level.store"
xmlns="http://www.ietf.org/xml/ns/mrcpv2">
<interpretation confidence="0.9">
<instance>
yes
</instance>
<input>yes</input>
</interpretation>
</result>
```

No campo `Completion-Cause`, o termo *sucess* (código `000`) indica um reconhecimento bem-sucedido. Outras alternativas comuns incluem *nomatch* (código `001`), caso a entrada de fala não corresponda a uma gramática ativa, e *no-input-timeout* (código `002`), caso a fala não tenha sido detetada antes do tempo limite. Os resultados de reconhecimento são contidos no corpo do evento `RECOGNITION-COMPLETE`. Neste exemplo, foi detetada a palavra *"yes"* com um grau de confiança de 90%.

Ao integrar estes protocolos de comunicação nos serviços de comunicação modernos, é possível simplificar e otimizar chamadas VoIP, videoconferências e transmissões de media em tempo real. Este processo resulta numa experiência de comunicação melhorada e segura tanto para os *providers* quanto para o utilizador comum.

2.4 Interfaces de Comunicação por Voz

Esta secção explora os elementos essenciais no processamento e na interação da fala humana, onde serão apresentadas tecnologias que desempenham papéis cruciais na facilitação da comunicação entre humanos e máquinas, abrangendo desde a interpretação e transcrição da fala até a conversão de texto em discurso audível.

2.4.1 *Automatic Speech Recognition*

ASR é um procedimento conduzido por sistemas computacionais para interpretar e transcrever a fala humana. Num sistema ASR convencional, os sinais sonoros são registados de um interlocutor por meio de um microfone, passando por uma análise usando padrões, modelos ou algoritmos específicos, produzindo uma saída geralmente em formato de texto [32].

A aplicação da tecnologia de processamento de fala é ampla. Em sistemas de mensagens unificadas, um subsistema de transcrição de fala pode converter mensagens de voz em texto, simplificando o envio e receção de mensagens em diversos formatos, como *e-mails* ou *Short Message Service* (SMS).

- **Assistentes Virtuais:** Tecnologias como a *Siri* da *Apple*, *Google Assistant*, *Amazon Alexa* e *Microsoft Cortana* utilizam ASR para compreender comandos de voz dos utilizadores e executar ações correspondentes, tais como responder a perguntas, definir lembretes e reproduzir música.

- **Centros de Atendimento Automatizado:** Muitas empresas empregam sistemas de IVR que utilizam ASR para reconhecer e processar as solicitações dos clientes por telefone. Os clientes podem, por exemplo, fornecer informações de conta, selecionar opções de menu ou aceder a serviços automatizados usando apenas a sua voz.

- **Transcrição Automática em Aplicações:** Aplicações de produtividade, como *Microsoft Office* e *Google Docs*, começam a integrar recursos de transcrição automática baseados em ASR. Isto permite aos utilizadores ditar texto em vez de o digitarem manualmente, poupando tempo e esforço na criação de documentos.

- **Comandos de Voz em Dispositivos Inteligentes:** Dispositivos inteligentes, como televisões, dispositivos de *streaming* e sistemas de automação residencial, incorporam frequentemente tecnologias de ASR que permitem aos utilizadores ter controlo sobre estes através de comandos de voz.

Estas aplicações refletem o potencial significativo da tecnologia de processamento de fala para melhorar a comunicação e a acessibilidade em diversos contextos.

Arquitetura Básica de um Sistema ASR

A arquitetura típica de um sistema ASR está ilustrada na Figura 2.14.

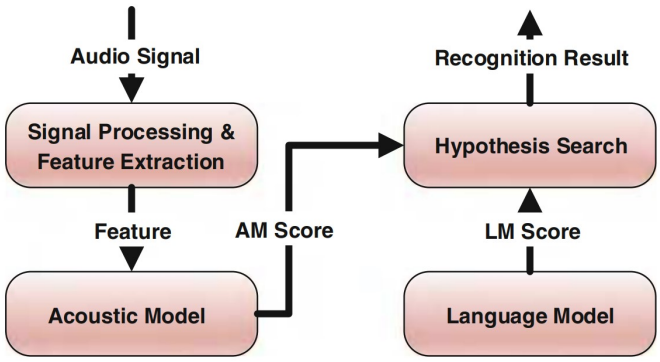


Figura 2.14: Arquitetura de um sistema ASR [33]

Conforme indicado na Figura 2.14, o sistema ASR tem quatro componentes principais: processamento de sinal e extração de características, modelo acústico, modelo de linguagem e procura de hipóteses. O componente de processamento de sinal e extração de características recebe como entrada um sinal de áudio, melhora a fala removendo ruídos e distorções de canal, converte o sinal do domínio do tempo para o domínio da frequência e extrai características que irão ser utilizados pelo modelo acústico. Este, por sua vez, integra conhecimento sobre acústica e fonética e utiliza as características geradas a partir do componente de extração para gerar uma pontuação de modelo acústico - pontuação AM - para a sequência de características de comprimento variável. O modelo de linguagem estima a probabilidade de uma sequência de palavras hipotética, ou pontuação de LM. Esta pontuação pode ser estimada com mais precisão se o conhecimento prévio sobre o domínio ou tarefa for conhecido. A componente de procura de hipóteses combina as pontuações de AM e LM dada a sequência de vetores de características e a sequência de palavras hipotetizadas, e produz a sequência de palavras com a maior pontuação como resultado de reconhecimento [33].

As duas principais questões a serem tratadas pelo componente de AM são os vetores de características de comprimento variável e a variabilidade nos sinais de áudio. A variabilidade nos sinais de áudio é causada pela interação complicada das características do falante, tais como o género ou doenças associadas, estilo e velocidade da fala, ruído, conversas paralelas, distorção de canal, diferenças de dialeto e sotaques não nativos. Um sistema de reconhecimento de fala bem-sucedido deve lidar com toda a variabilidade acústica.

2.4.2 *Text-To-Speech*

O processamento digital de fala desempenha um papel crucial na pesquisa e nas aplicações contemporâneas de comunicação por voz. O sistema TTS é responsável por transformar texto em voz, utilizando um sintetizador de fala [34]. A principal finalidade deste sistema é converter um texto fornecido arbitrariamente numa forma de onda falada correspondente.

Componentes de um Sistema TTS

Podemos dividir um sistema TTS tal como demonstra a Figura 2.15.

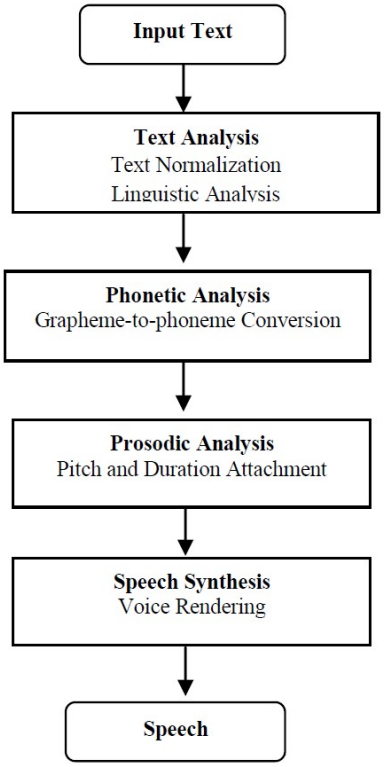


Figura 2.15: Diagrama de blocos de um sistema TTS [35]

Estes procedimentos podem ser resumidos em dois componentes fundamentais: o processamento de texto e a síntese de fala.

- **Processamento de texto:** Nesta fase, o texto é analisado linguisticamente para compreender a sua estrutura gramatical e semântica. Isso envolve etapas como análise fonética, morfológica, sintática e semântica, visando interpretar o conteúdo textual de forma precisa e abrangente. Além disso, técnicas de pré-processamento podem ser aplicadas para normalizar o texto, corrigir erros ortográficos e segmentar o conteúdo em unidades menores para uma manipulação mais eficiente.

- **Síntese de Fala:** Após o processamento do texto, a síntese de fala entra em ação para transformar as informações interpretadas num discurso audível. Esta etapa envolve a seleção e concatenação de unidades de fala, como fonemas ou palavras, para construir a saída sonora correspondente ao texto de entrada. Técnicas avançadas de síntese de fala, como síntese por concatenação ou síntese por regras, são aplicadas para garantir uma produção de fala natural e fluente.

Estes dois componentes trabalham em conjunto para possibilitar a conversão eficiente de texto em fala, desempenhando um papel crucial na comunicação por voz e na acessibilidade para diversos utilizadores.

Capítulo 3

WMS Adapter

Este capítulo tem como objetivo apresentar a estrutura e evolução do `WMS Adapter`. Além disso, serão discutidas as funcionalidades acrescidas ao `WMS Adapter` ao longo do projeto, destacando como estas implementações contribuíram para a sua melhoria.

3.1 Apresentação do `WMS Adapter`

Esta secção descreve a transição de uma configuração inicial mais simples para uma solução mais robusta e eficiente, destacando o papel crucial do `WMS Adapter` nesta evolução.

Atualmente, a arquitetura de referência do sistema é composta por dois componentes principais, o Asterisk-i e o WMS. O Asterisk-i é uma versão adaptada pela Altice Labs que tem como base a versão 13 do Asterisk. Esta versão surgiu da necessidade específica da Altice Labs de obter funcionalidades que não eram disponibilizadas pelo Asterisk e que não estavam previstas para desenvolvimento futuro do mesmo.

De forma a integrar estes dois componentes, foi desenvolvida uma aplicação integrada no Asterisk-i que comunica com o WMS através da troca estruturas binárias por meio de um *socket* de comunicação, conforme ilustrado na Figura 3.1. Como o desenvolvimento das funcionalidade necessárias no Asterisk implicava a implementação direta sobre o *core* do Asterisk e, devido ao elevado número de alterações sobre o mesmo, acabou por tornar a migração para novas versões do Asterisk cada vez

mais complexa, implicando muito esforço por parte da equipa de desenvolvimento para garantir o correto funcionamento das implementações previamente realizadas.

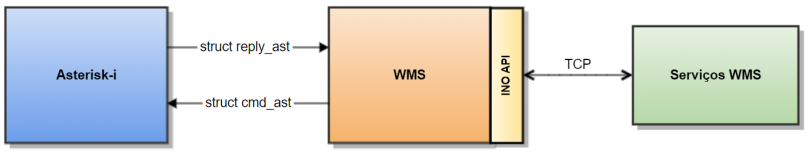


Figura 3.1: Versão inicial do projeto

Com o avanço do projeto, surgiu a necessidade de evoluir a versão do Asterisk com o menor impacto possível ao WMS, o que levou ao desenvolvimento de uma aplicação intermediária por parte da Altice Labs denominada `WMS Adapter`, conforme demonstrado na Figura 3.2.

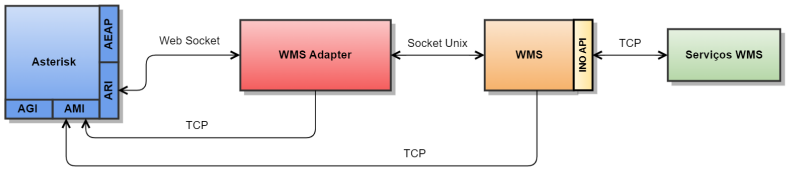


Figura 3.2: Versão final do projeto

Esta aplicação, desenvolvida em linguagem `Go`, para além de ter a finalidade de servir como ponte entre o Asterisk e o WMS, também é dotada da capacidade de criação de serviços independentes do uso do WMS como demonstrado na Figura 3.3.

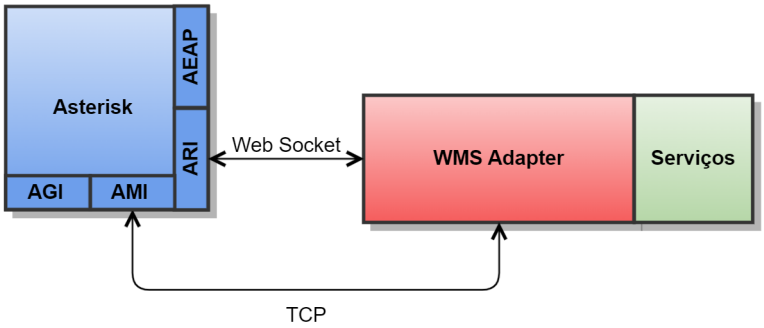


Figura 3.3: `WMS Adapter` com serviços

Na integração com o WMS, o `WMS Adapter` implementa o mesmo protocolo usado entre o WMS e o Asterisk-i, garantindo uma migração de arquitetura o mais transparente possível. Este protocolo de comunicação consiste na troca de estruturas binárias (`cmd_ast` e `reply_ast`) entre estas duas entidades através de um *socket*

UNIX. Estas estruturas binárias contêm cópias de informações essenciais das estruturas do WMS, necessárias para o processamento dos comandos `InoAPI` e suas respetivas respostas. Estes comandos são a forma pela qual o WMS comunica com os seus serviços, facilitando a execução de funções e a troca de informações.

Quanto à criação de serviços utilizando o `WMS Adapter`, estes apresentam uma arquitetura simplificada relativamente à arquitetura de integração com o WMS, sendo que, todas as funcionalidades que o WMS usa através do `WMS Adapter`, este tipo de serviços também consegue tirar partido.

Com o desenvolvimento do `WMS Adapter`, pretende-se facilitar a migração para versões mais recentes do Asterisk, o que acaba por fornecer novas funcionalidades, correções de *bugs* e melhorias de segurança.

Estrutura do `WMS Adapter`

Nesta secção, iremos analisar a estrutura do `WMS Adapter`, destacando os principais ficheiros e pacotes que compõem o seu funcionamento. O código do `WMS Adapter` está dividido da seguinte maneira:

- **`main.go`:** O ficheiro `main.go` serve como ponto de entrada do `WMS Adapter`. Aqui é criada uma instância do `StasisManager`, responsável por gerir a interação com o Asterisk. Neste ficheiro é também estabelecido um *loop* para processar os eventos recebidos do Asterisk e encaminhá-los para tratamento posterior no `event.go`.

- `event.go` O ficheiro `event.go` contém a lógica para processamento de eventos provenientes do Asterisk e WMS. Este é responsável por interpretar os diferentes tipos de eventos recebidos relativamente a chamadas, como a sua entrada e saída da `Stasis`, chamada recebida ou efetuada, entre outras, tomando as decisões apropriadas com base nessas informações. Além disso, é também responsável por todos os comandos enviados pelo WMS, tais como dar início a conferências, toques de anúncios ou conversão de texto em voz e vice-versa.

- **Ficheiros de pacotes:** Esta categoria engloba os ficheiros que constituem os pacotes `pkg` e `internal/pkg`. Aqui reside todo o código responsável por fornecer as funcionalidades essenciais para a interação do `WMS Adapter` com o Asterisk e WMS. Estes pacotes são utilizados para agrupar funcionalidades e componentes relacionados, permitindo uma organização lógica do código-fonte, facilitando o desenvolvimento, manutenção e evolução do `WMS Adapter` a longo prazo.

3.2 Requisitos

Como melhorias do `WMS Adapter`, foi requisitado adicionar suporte a chamadas de saída e implementar a capacidade de realizar *route endpoints*. Todo o processo de implementação e integração com aplicações em `Go` está representado neste capítulo.

3.2.1 Chamadas de Saída

A primeira fase do projeto incide em adicionar suporte a chamadas de saída (*outbound*) ao `WMS Adapter`, onde até então suportava apenas chamadas de entrada (*inbound*). Esta funcionalidade pode ser útil em cenários como *telemarketing* ou qualquer situação em que o `WMS Adapter` precise de iniciar comunicação com o cliente.

A distinção entre estes dois tipos de chamadas reside no facto de que uma chamada de entrada tem como destino o Asterisk, enquanto uma chamada de saída é originada pelo próprio Asterisk.

Esta expansão do `WMS Adapter` requer modificações significativas no código e na lógica de funcionamento. Para conseguir testar o fluxo de chamadas do `WMS Adapter`, será necessário recorrer a um *softphone* para fazer e receber chamadas VoIP. Para tal, foi utilizado o *software* PhonerLite [36].

Implementação

O primeiro passo consiste em distinguir a informação proveniente do Asterisk das chamadas recebidas e efetuadas. Para alcançar essa distinção, utilizamos a ferramenta de terminal `sngrep`. Através desta, é possível observar a diferença entre uma chamada de entrada, exemplificada na Listagem 3.1, e uma chamada de saída, demonstrada na Listagem 3.2.

```
INVITE sip:888@10.113.137.250 SIP/2.0
Via: SIP/2.0/UDP 10.112.208.195:5060;branch=
    z9hG4bK8062ace1b787ee11a0028c6b03e9b186;rport
From: <sip:Goncalo@10.113.137.250>;tag=47636891
To: <sip:888@10.113.137.250>
Call-ID: 8062ACE1-B787-EE11-A001-8C6B03E9B186@10.112.208.195
CSeq: 1 INVITE
```

Listagem 3.1: `INVITE` de uma chamada de entrada

```
INVITE sip:4433@10.112.208.195:5060 SIP/2.0
Via: SIP/2.0/UDP 10.113.137.250:5060;rport;branch=
    z9hG4bKPj50abae1a-7abf-4bae-a822-d7d6282468da
From: <sip:goncalo@198.18.173.164>;tag=19a88304-0679-4e2c-a5e7-6
    debb3ae2ffa
To: <sip:4433@10.112.208.195>
```

```
Contact: <sip:asterisk@10.113.137.250:5060>
Call-ID: 68628f31-e465-42a7-b9ea-610d8e015141
CSeq: 22208 INVITE
```

Listagem 3.2: `INVITE` de uma chamada de saída

Numa chamada de entrada, o `INVITE` é redirecionado para o IP do servidor do Asterisk (10.113.137.250), ao contrário das chamadas de saída, onde é encaminhado para o IP da máquina onde está em execução o *softphone* (10.112.208.195). O objetivo principal é habilitar o `WMS Adapter` a distinguir entre os dois tipos de chamadas, possibilitando assim o seu processamento de forma distinta. Esta diferenciação é fundamental para garantir a correta execução das funcionalidades do `WMS Adapter` em diversos contextos de comunicação.

Para tal, foi necessário observar a informação recebida pelo `WMS Adapter` relativamente ao WMS, tal como demonstra a Listagem 3.3.

```
%! (EXTRA *ari.StasisStart=&
{
    {
        WMSAPP fa:16:3e:8d:6c:25 2023-11-23 09:56:05.52 +0000 WET
    StasisStart
    }
    map[] [36 1 3 500]
    {
        <nil>
        1700733359.0
        PJSIP/default-00000000
        Up
        <goncalo>
        <goncalo>
        2023-11-23T09:55:59.665Z
        context: "default"
        exten: "s"
        priority:1
        map[]
        {}
        []
        0
    }
}
```

Listagem 3.3: Logs de uma chamada de saída

A principal diferença entre estas duas chamadas reside no facto de que na mensagem recebida de uma chamada de entrada está inserido o parâmetro `Ring`, enquanto na chamada de saída está inserido o parâmetro `Up`. Através desta variação, o `WMS Adapter` poderá diferenciar entre os dois tipos de chamadas.

Para implementar esta funcionalidade, foi necessário fazer alterações no ficheiro `event.go`. Foi necessário implementar esta distinção na função que recebe os eventos da ARI, tal como demonstra a Listagem 3.4.

```
else if channel.date.GetState() == "Up" {
    event_processor_logger.Debug("Outbound Call")
    return processOutboundCallReceived(client,container,ev)
} else {
     event_processor_logger.Debug("Inbound Call")
     return processStasisStartCallReceived(client,container,ev)
}
```

Listagem 3.4: Diferenciação entre chamadas de saída e entrada

Caso o estado da chamada seja `Ring`, ela será considerada uma chamada de entrada e será encaminhada para a função responsável pelo tratamento dessas chamadas. Por outro lado, se o estado da chamada for `Up`, será interpretada como uma chamada de saída e será direcionada para uma função capaz de lidar com este tipo de chamadas.

Nesta função é feito inicialmente uma distinção entre uma chamada de saída executada pelo WMS (se houver um argumento associado ao evento de início de `Stasis`), ou se foi realizada por um serviço externo, discutido na seção 3.2.1. Em ambos os casos, são extraídos os dados do canal e os argumentos associados.

No caso de ser uma chamada originada pelo WMS, a função `processOutboundReceivedCallData()` é chamada para armazenar todos os dados relativos à execução da chamada, a fim de fornecer informações adicionais ao utilizador sobre a chamada realizada. Quando esta função é invocada, o evento `OUTBOUND_CALL_ACCEPTED` é enviado para a `Stasis` de modo a informar o utilizador do acontecimento.

Para demonstrar o bom funcionamento das chamadas de saída, foi desenvolvido um serviço com o único objetivo de realizar apenas este tipo de chamadas. A execução deste serviço garante que o sistema é capaz de lidar adequadamente com as chamadas de saída, mesmo sem a intervenção direta do WMS.

Criação de Serviços Utilizando o `WMS Adapter`

Foi criado uma aplicação onde é pedido ao utilizador para digitar o endereço IP do utilizador para o qual deseja fazer a ligação, bem como o nome que irá aparecer no ecrã ao realizar essa chamada. Após a entrada destes parâmetros, a chamada é realizada.

Para tal, foi necessário desenvolver uma função que verifica se o gerenciador é válido, configura a requisição de chamada com o destino e tenta então iniciar a chamada com recurso à ARI. A função garante que o sistema consegue realizar chamadas de saída de forma eficaz, sendo esta uma parte crucial para a validação da gestão de chamadas no sistema. Para uma melhor compreensão do funcionamento do serviço de chamadas de saída, é importante analisar o fluxo de mensagens SIP envolvido no processo. A Figura 3.4 ilustra o fluxo de mensagens SIP que ocorre durante uma chamada de saída.

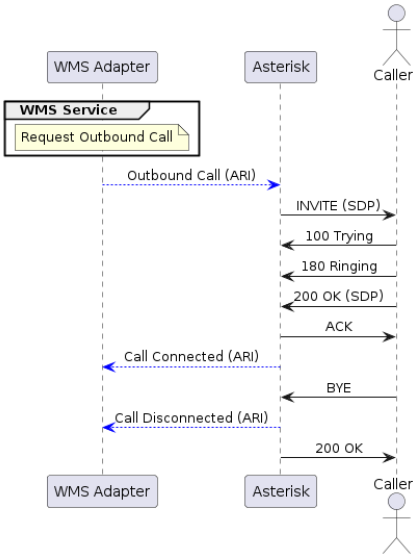


Figura 3.4: Diagrama de sequência de uma chamada de saída

Primeiramente, o serviço solicita ao `WMS Adapter` a realização de uma chamada de saída, que irá fornecer um evento determinado através da ARI. Em seguida, o Asterisk inicia uma chamada SIP para o destino indicado pelo serviço. Tanto para o caso de atendimento com sucesso (Figura 3.4) da chamada como para casos de insucesso, o Asterisk notifica o `WMS Adapter` com o evento correspondente via ARI. Quando o chamador decide encerrar a chamada, ele envia uma mensagem "`BYE`". Este evento é processado pelo Asterisk, que então envia uma notificação de chamada desconectada ao `WMS Adapter` através da ARI, finalizando assim a chamada.

3.2.2 *Route Endpoint*

A segunda melhoria a implementar no `WMS Adapter` consiste em fornecer suporte para a realização de *route endpoints* entre duas chamadas. Esta funcionalidade permite que o `WMS Adapter` conecte duas chamadas através de um *endpoint* específico. O *endpoint* de uma chamada representa o ponto de conexão que define o destino ou origem da chamada dentro da rede de comunicação. Este ponto de conexão pode corresponder a um utilizador, grupo, serviço ou dispositivo específico.

Implementação

Para implementar este recurso no `WMS Adapter`, foi necessário analisar o comportamento do WMS quando este requisitava um *route endpoint* entre duas chamadas. Este pedido está representada na Listagem 3.5.

```
con_id: 36
cmd_cnt: 103
cmd_id: 9
edp_id: 33
clear_dig_buf: 0
ch_index: 10
digits: 4433@10.112.208.255:5060
dtmf: 0
asr: 0
al_int_val1: -1
al_int_val2: -1
al_long_val1: 0
al_long_val2: 0
al_char_val1:
al_char_val2:
prompt_id:
use_smra:
extradata:
```

Listagem 3.5: Logs de um pedido de *route endpoint*

Neste extrato, é possível observar vários parâmetros cruciais para o estabelecimento de um *route endpoint*. Alguns dos parâmetros mais importantes incluem:

- **con_id:** Identificador da conexão, único para cada chamada.
- **cmd_id:** Identificador do comando, cujo valor identifica um pedido de *route endpoint*.
- **edp_id:** Identificador do *endpoint*, essencial para identificar a chamada a ser processada.
- **digits:** O número digitado e o respetivo endereço IP associado, que indica para onde a chamada está a ser encaminhada.

Com base nestes dados, o `WMS Adapter` pode tomar decisões sobre como processar e encaminhar chamadas. Para implementar esta funcionalidade, foi necessário seguir uma série de etapas para garantir o seu correto funcionamento.

De acordo com a informação proveniente do WMS, o `WMS Adapter` recebe uma variedade de comandos essenciais para executar diversas funcionalidades presentes no sistema. Estes comandos representam a principal forma de comunicação do WMS com as aplicações que fazem uso do mesmo. Entre os comandos disponíveis estão aqueles associados ao pedido de execução de determinadas tarefas, tais como pedidos de TTS, ASR, *route endpoint*, chamadas de saída e entrada, entre outros.

O trecho de código apresentado na Listagem 3.6 descreve a ação que o `WMS Adapter` deve realizar ao detetar o comando `ROUEND`, responsável pelo *route endpoint*.

```
confId := fmt.Sprint(cmd_ast.Edp_id)

wms_event_processor_logger.Info("Creating Conference", "confId",
    confId)

manager.Routendpoint(_channel, _channel2, confId, rec_file)
```

Listagem 3.6: Extrato de código da função `processWMSCommands()`

Neste excerto é gerado um identificador único para a conferência com base no *endpoint* associado à chamada, seguindo-se o registo do início do processo de criação da conferência com o seu identificador. É também invocada a função `Routendpoint()` com os parâmetros relacionados aos canais em que será feito a conferência, bem como o seu ID e, se necessário, o nome do ficheiro em qual irá ser feita a gravação da chamada.

Dentro da função `Routendpoint()` verifica-se se ambos os canais não têm *route endpoints* associados. Caso se confirme, são conectados entre si, estabelecendo uma conferência com um identificador único gerado a partir do ID do *endpoint*. Caso já exista uma conferência ativa entre um dos canais, a função regista uma mensagem informativa e retorna sem fazer alterações. Após a criação bem-sucedida da conferência, os participantes são adicionados à conferência. Por fim, se um ficheiro de gravação tiver sido especificado, a conferência é configurada de modo a gravar a chamada utilizando como nome do ficheiro o parâmetro `rec_file` fornecido.

Após realizar com sucesso um *route endpoint* entre duas chamadas, é possível observar no Asterisk as conferências ativas através do comando `bridge show all`. Um exemplo de uma conferência está representada na Figura 3.5.

```
Connected to Asterisk 20.6.0 currently running on 514efdd87dd4 (pid = 26)
Unable to read or write history file '/root/.asterisk_history'
514efdd87dd4*CLI> bridge show all
Bridge-ID                           Chans Type            Technology      Duration
rouend_10                               2 stasis          simple_bridge   00:01:18
```

Figura 3.5: Lista de conferências ativas no Asterisk

Tal como referido anteriormente, ao realizar uma conferência, o seu identificador único será gerado com base no *endpoint* de uma das chamadas, ficando neste caso com o nome `rouend_10`, sendo também possível observar a presença de dois participantes, identificado pelo número do parâmetro `Chans`.

Criação de Serviços Utilizando o `WMS Adapter`

De forma similar ao que foi realizado com as chamadas de saída, foi desenvolvido um serviço com o intuito de demonstrar o bom funcionamento da implementação do requisito *route endpoint*. Ao iniciar este serviço, o serviço aguarda por uma chamada de entrega. Após esta ser atendida, o serviço disputa de forma autónoma uma

chamada de saída. Uma vez atendida, é realizada a operação de *route endpoint* cujo objetivo é colocar em conversação as duas chamadas, assim como se pode verificar no excerto de código apresentado na Listagem 3.7.

```
case event.CALL_ACCEPTED:
  {
        var rec_file string = ""
        var confId string = "20"

            if hasEnteredCase {
                event_processor_logger.Info("Creating Conference")
                manager.Routendpoint(_channel, _channel2, confId,
                rec_file)
                event_processor_logger.Info("Conference Created")
            }
        hasEnteredCase = true
        _channel2 = _channel
  }
```

Listagem 3.7: Extrato de código do serviço de amostra *route endpoint*

O extrato de código do serviço apresentado demonstra apenas o evento `CALL_ACCEPTED`, dado que este controla a invocação do *route endpoint*. Esta invocação é realizada apenas após a receção do segundo evento `CALL_ACCEPTED` que, no caso deste serviço, corresponde à chamada de saída. De forma a conectar as duas chamadas, a informação transmitida para o *route endpoint* pertence aos dois canais associados às chamadas previamente estabelecidas e ao identificador da *bridge* que se irão conectar. Para este serviço, o identificador da *bridge* escolhido foi o 20. Após realizada esta operação, as duas chamadas estabelecidas isoladamente passam a comunicar entre si.

A Figura 3.6 apresenta uma possível representação de um diagrama de sequência para este caso.

Inicialmente, uma chamada é realizada para o Asterisk (`INVITE`) pelo interveniente *Caller 1*. O evento disputado pelo Asterisk é enviado para o `WMS Adapter` através da ARI, onde esta notifica o serviço. Por sua vez, o serviço dá instrução para a chamada ser aceite e requisita uma chamada de saída. Quando esta é atendida pelo interveniente *Caller 2*, o `WMS Adapter` faz um novo pedido para a realização de *route endpoint*, onde se reflete em três operações para o Asterisk (uma para a criação da *bridge* e duas para a colocação dos canais dos intervenientes). Por fim, o *Caller 2* desliga a chamada assim que o `WMS Adapter` é notificado e este instrui o Asterisk para a remoção dos canais da *bridge* e eliminação da *bridge* correspondente.

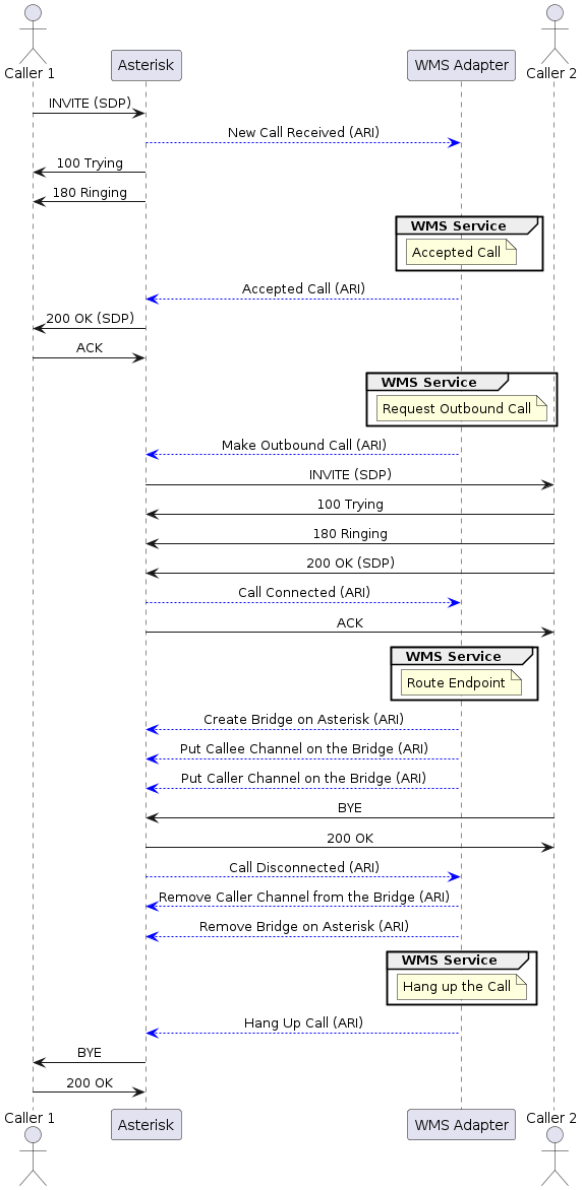


Figura 3.6: Diagrama de sequência de um *route endpoint*

Capítulo 4

Invocar Aplicações *Dialplan* do Asterisk

Este capítulo tem como objetivo apresentar as abordagens para invocar aplicações *Dialplan* do Asterisk, assim como a implementação da abordagem mais adequada.

4.1 Estudo das Abordagens

Para integrar aplicações *Dialplan* no `WMS Adapter` foi necessário analisar diversas abordagens de implementação. Esta análise foi fundamental para entender as vantagens e limitações de cada método e escolher a solução mais adequada para as necessidades do projeto.

4.1.1 Invocação Direta no *Dialplan*

Como primeira abordagem, foi estudado a possibilidade de invocar diretamente aplicações através do *Dialplan*. Esta abordagem permitiria uma implementação simples e direta, utilizando a linguagem nativa do Asterisk para controlar o fluxo das chamadas e invocar as aplicações necessárias. No entanto, o maior obstáculo desta abordagem é seu caráter estático: sempre que for necessário chamar uma das diversas aplicações *Dialplan* do Asterisk, será necessário editar manualmente o *Dialplan* e selecionar a aplicação desejada. Um exemplo desta abordagem com a aplicação `Playback` está representada na Listagem 4.1.

```
[default]
exten => _[a-zA-Z0-9=].,1,Set(INVITE_TO=${PJSIP_HEADER(read,To)})
same => n,Set(INVITE_FROM=${PJSIP_HEADER(read,From)})
same => n,Set(INVITE_URI=${CHANNEL(pjsip,request_uri)})
same => n,Set(CALL_ID=${PJSIP_HEADER(read,Call-ID)})
same => n,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},${INVITE_URI},
${CALL_ID})
same => n,Playback(hello-world)
same => n,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},${INVITE_URI},
${CALL_ID})
exten => h,1,Hangup
```

Listagem 4.1: Exemplo de uma invocação direta no *Dialplan*

Esta necessidade de edição manual pode ser demorada e propensa a erros, especialmente em ambientes onde as necessidades das chamadas mudam com frequência. Além disso, a sua falta de flexibilidade torna esta abordagem inadequada para cenários mais complexos ou para aqueles que exigem uma integração contínua com outras aplicações ou sistemas externos.

Devido a esta limitação, a abordagem de invocação direta no *Dialplan* foi descartada em favor de soluções mais dinâmicas e flexíveis, que permitam uma maior automatização e integração com o sistema como um todo.

4.1.2 Invocação Através da AGI no *Dialplan*

Outra abordagem considerada foi a utilização da AGI no *Dialplan* para invocar aplicações utilizando ficheiros de configuração. Semelhante à abordagem direta, utiliza também o *Dialplan* mas invoca uma API capaz de lidar com as aplicações desejadas.

Contudo, a invocação através da AGI ainda mantém um caráter estático e manual. Sempre que uma nova aplicação precisa ser executada, é necessário editar o *Dialplan*. Um exemplo desta abordagem com a aplicação `Playback` está representada na Listagem 4.2.

```
[default]
exten => _[a-zA-Z0-9=].,1,Set(INVITE_TO=${PJSIP_HEADER(read,To)})
same => n,Set(INVITE_FROM=${PJSIP_HEADER(read,From)})
same => n,Set(INVITE_URI=${CHANNEL(pjsip,request_uri)})
same => n,Set(CALL_ID=${PJSIP_HEADER(read,Call-ID)})
same => n,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},${INVITE_URI},
${CALL_ID})
same => n,AGI(Playback(hello-world))
same => n,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},${INVITE_URI},
${CALL_ID})
exten => h,1,Hangup
```

Listagem 4.2: Exemplo de uma invocação através da AGI no *Dialplan*

Esta abordagem leva a problemas semelhantes aos da invocação direta. Devido a esta limitação, a abordagem de invocação através da AGI no *Dialplan* foi descartada.

4.1.3 Invocação Através de um *Wrapper* AGI

De acordo com as abordagens mencionadas anteriormente, era necessário encontrar uma solução dinâmica para utilizar as aplicações sem a constante necessidade de alterar o *Dialplan* do Asterisk. Para tal, foi estudado a possibilidade de utilizar um *wrapper* AGI [37]. Esta abordagem está representada na Figura 4.1.

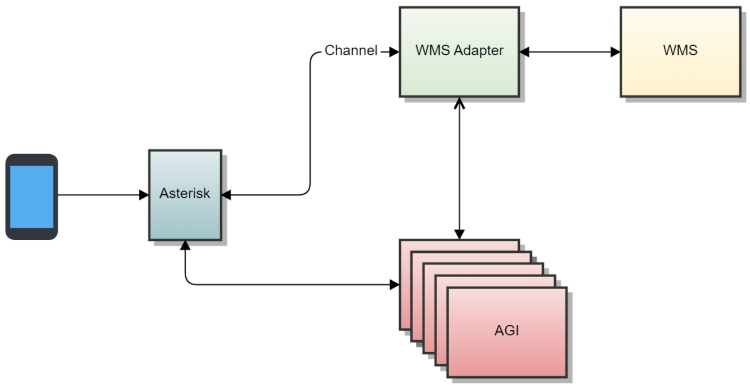


Figura 4.1: Diagrama de blocos da abordagem *wrapper* AGI

Utilizando este *wrapper*, é possível invocar aplicações *Dialplan* de forma dinâmica através do `WMS Adapter`. Esta solução permite uma maior flexibilidade, pois elimina a necessidade de edições manuais constantes no *Dialplan*, permitindo que as chamadas às aplicações sejam geridas de maneira mais automatizada.

No entanto, esta abordagem apresenta algumas desvantagens. O principal problema surge quando o canal está fora do `WMS Adapter`. Caso o WMS envie informações para esse canal, todos os dados serão perdidos, uma vez que o canal não estará mais sob o controlo direto da `Stasis`. Além disso, cada vez que a AGI é invocada, um novo processo é iniciado. Isto pode resultar num aumento considerável no uso de recursos do sistema, impactando a performance em ambientes com alta demanda de chamadas. Outro problema é o facto de nenhum evento ser transmitido para o `WMS Adapter`, uma vez que a AGI por natureza não os envia.

Apesar das melhorias, a invocação através de um *wrapper* AGI apresentou algumas limitações, o que levou esta abordagem a ser descartada.

4.1.4 Invocação Através de um *Wrapper* AMI

Na procura de uma solução mais eficiente, foi adotado o uso de um *wrapper* AMI [38] com o intuito de invocar aplicações *Dialplan*. Contudo, nem todas as aplicações estão acessíveis via AMI. Para contornar esta limitação, a solução encontrada foi utilizar um *wrapper* que pudesse invocar não só as aplicações acessíveis pela AMI, mas também aquelas que estão disponíveis através da AGI. Com o *wrapper* AMI, conseguimos ultrapassar esta limitação uma vez que a AMI tem a capacidade de invocar a própria AGI.

A integração destas duas APIs com a ARI permite um controle mais abrangente do Asterisk conforme ilustrado na Figura 4.2.

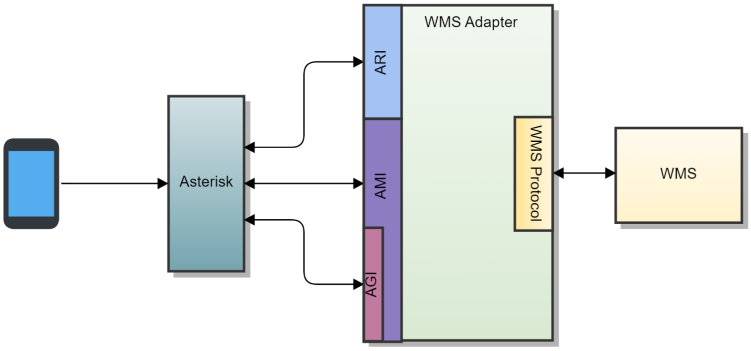


Figura 4.2: Diagrama de blocos da abordagem *wrapper* AMI

Uma vantagem significativa desta abordagem é que, embora o `WMS Adapter` não receba eventos diretamente da AGI, recebe eventos da AMI, o que proporciona um maior controlo sobre o estado das chamadas.

4.1.5 Avaliação das Abordagens

Um resumo destas abordagens está representado na Tabela 4.1.

Requisitos/Pontos de avaliação	**WMS Adapter + Dialplan (Invocar aplicações diretamente)**	**WMS Adapter + AGI (Dialplan)**	**WMS Adapter + AGI (wrapper)**	**WMS Adapter + AMI (wrapper) + AGI (Invocar a AGI através da AMI)**
É uma solução dinâmica?	×	×	✓	✓
É possível invocar aplicações através do `WMS Adapter` diretamente?	×	×	×	×
O `WMS Adapter` continua a ser notificado com DTMFs quando liberta o canal para o *Dialplan*?	—	—	×	×
Recebe eventos no `WMS Adapter` mesmo quando liberta o canal para o *Dialplan*?	—	—	⚠ Não recebe eventos de aplicações invocadas pela AGI	⚠ Recebe eventos de aplicações invocadas pela AMI, mas não da AGI
É possível ter um canal no `WMS Adapter` e ser controlado pela AGI?	×	×	×	×
É possível ter controlo total da chamada?	×	×	⚠ Apenas quando a aplicação não é *blocker*	⚠ Apenas quando a aplicação não é *blocker*
É necessário ativar a AGI no *Dialplan*?	—	—	✓	✓
Tem acesso a funcionalidades da AMI e AGI?	—	—	⚠ Apenas tem acesso a funcionalidades da AGI	✓
Ao invocar as aplicações *Dialplan* do Asterisk, todas podem ser paradas e/ou terminadas?	—	—	×	×

Tabela 4.1: Comparação entre abordagens de integração `WMS Adapter` e aplicações *Dialplan* do Asterisk

Apesar das melhorias previamente mencionadas, é importante realçar que algumas limitações persistem, como a incapacidade de receber eventos diretamente da AGI e a perda parcial de controlo sobre a chamada quando a aplicação desejada é *blocker*. No entanto, estas desvantagens são compensadas pelos vários benefícios proporcionados pelo uso desta última abordagem. Tendo em consideração estes aspetos, foi decidido implementar o *wrapper* AMI devido ao facto de ser a solução mais eficaz para a invocação de aplicações *Dialplan* do Asterisk no `WMS Adapter`.

4.2 Implementação do *Wrapper* AMI

O propósito desta implementação é permitir que a aplicação `WMS Adapter` consiga invocar aplicações *Dialplan* do Asterisk via AMI. No entanto, uma vez que nem todas as aplicações estão acessíveis através da AMI e a AGI possui uma função que permite a invocação de qualquer aplicação, a implementação passa por usar a AGI através do *wrapper* AMI. De modo a implementar a abordagem relativa ao *wrapper* AMI, foi necessário fazer algumas alterações, tanto ao *Dialplan* do Asterisk, assim como ao `WMS Adapter`.

4.2.1 Construção do *Dialplan*

Tendo em conta a necessidade de invocar aplicações via AGI, foi necessário criar um contexto no Asterisk dedicado a esta funcionalidade como representado na Listagem 4.3. Este novo contexto tem o propósito de invocar qualquer aplicação *Dialplan* do Asterisk.

```
[ExecAst]

exten => ExecAst,1,AGI(agi:async)
```

Listagem 4.3: Contexto `ExecAst` do *Dialplan*

A extensão apresentada indica ao Asterisk para executar a *Async* AGI, um método mais recente de utilizar a API. Este permite que uma aplicação AMI possa ser usada para controlar chamadas utilizando comandos AGI. Esta abordagem é útil para quem já faz uso da AMI e deseja aproveitar a mesma aplicação para controlar chamadas [5].

Para ilustrar o uso prático dessa configuração, o contexto `ExecAst` pode ser integrado juntamente com o contexto *default* no *Dialplan* do `WMS Adapter` tal como demonstrado na Listagem 4.4.

```
[default]

exten => _[a-zA-Z0-9=].,1,Set(INVITE_TO=${PJSIP_HEADER(read,To)})
same => n,Set(INVITE_FROM=${PJSIP_HEADER(read,From)})
same => n,Set(INVITE_URI=${CHANNEL(pjsip,request_uri)})
same => n,Set(CALL_ID=${PJSIP_HEADER(read,Call-ID)})
same => n,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},${INVITE_URI},
${CALL_ID})
exten => s,1,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},
${INVITE_URI},${CALL_ID})
exten => h,1,Hangup
```

Listagem 4.4: Contexto de invocação de `Stasis` do *Dialplan*

O contexto demonstrado acima representa o fluxo normal de uma chamada que entra no `WMS Adapter`. Quando uma chamada é recebida, são capturados os cabeçalhos SIP, URI e o ID da chamada. A chamada entra na aplicação `Stasis`, onde todo o processamento da chamada é executado de acordo com as instruções configuradas no `WMS Adapter`.

4.2.2 Integração da funcionalidade ExecAst

Quando o utilizador solicita a invocação de uma aplicação, o evento `EXEAST` é recebido pelo `WMS Adapter` e a função `ExecAst()` é executada. Esta função é responsável por executar a aplicação desejada pelo utilizador, podendo ser dividida em duas partes. A primeira faz referência à mudança de contexto criado. Esta mudança é importante pois é necessário sair da `Stasis` para executar aplicações, uma vez que o Asterisk não suporta a sua invocação dentro da mesma. Esta mudança de contexto está representada na Listagem 4.5.

```
context := "ExecAst"
extension := "ExecAst"
priority := 1

err := _chan.Continue(context, extension, priority)
```

Listagem 4.5: Código do contexto `ExecAst` do *Dialplan*

Através da função `Continue()` da ARI, é possível mover um determinado canal para um contexto e extensão específicos. Aqui, o canal é redirecionado para o contexto `ExecAst` previamente representado na Listagem 4.3. No `WMS Adapter`, iremos utilizar a aplicação *Dialplan* `Exec`, que possibilita a invocação de qualquer aplicação, mesmo que não esteja listada nas aplicações acessíveis diretamente via AMI [39].

A segunda parte da função `ExecAst()` realiza exatamente isso. Nesta função é inicialmente configurada uma variável com o nome da aplicação desejada e os parâmetros fornecidos pelo utilizador. Esta variável é então enviada para que a AGI execute a aplicação especificada através da interface AMI. A chamada a esta função é realizada duas vezes: a primeira para executar a aplicação desejada e a segunda para retornar ao contexto original, o correspondente à aplicação `Stasis`, assegurando que o controlo da chamada seja retomado pela mesma após a execução da aplicação *Dialplan*. Este mecanismo oferece uma flexibilidade significativa, permitindo a invocação dinâmica de qualquer aplicação de *Dialplan*.

Após a troca de contexto, podemos observar o estado do canal para entender onde se encontra no sistema. A Figura 4.3 demonstra a localização do canal quando este está fora da `Stasis` a executar a aplicação `VoiceMail`, com o parâmetro adicional `1234` que representa a caixa de correio (*mailbox*) para a qual a mensagem foi enviada.

```
Channel                     Location                State   Application(Data)
PJSIP/anonymous-0000000e    ExecAst@ExecAst:0       Up      VoiceMail(1234)
```

Figura 4.3: Estado do canal durante um `ExecAst`

O diagrama de sequência na Figura 4.4 demonstra todo o processo de invocação da aplicação `VoiceMail` no *Dialplan* do Asterisk.

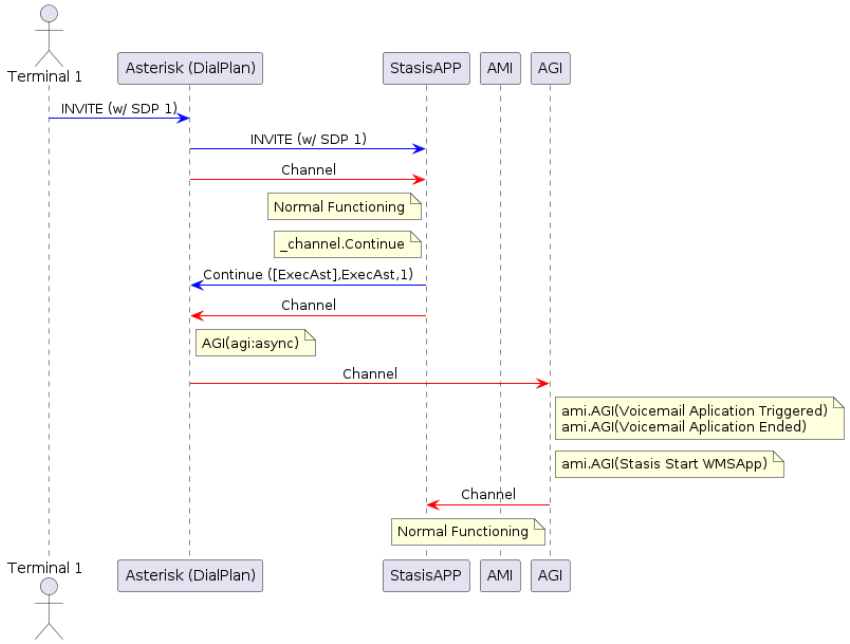


Figura 4.4: Diagrama de sequência da abordagem AMI *wrapper*

Como referido anteriormente, para executar a aplicação `VoiceMail` é necessário alterar temporariamente o fluxo da chamada para o contexto `ExecAst` do *Dialplan*. Neste contexto, a AGI é invocada mediante o *wrapper* da AMI. Esta alteração é realizada através do comando `_channel.Continue`, que instrui o Asterisk a prosseguir a execução do *Dialplan* no contexto `ExecAst`, executando a extensão com o mesmo nome. Dentro deste contexto, a chamada é direcionada para a aplicação `Exec`, que por sua vez invoca a aplicação. Após a sua execução, o controlo da chamada retorna ao contexto original no *Dialplan*, onde a aplicação `Stasis` continuará a gerir a chamada.

A implementação da invocação de aplicações *Dialplan* no Asterisk através do *wrapper* AMI demonstrou ser a abordagem mais eficaz e flexível para o `WMS Adapter`. Ao permitir a integração dinâmica das aplicações sem a necessidade de modificações constantes no *Dialplan*, esta solução não só melhora a eficiência operacional como proporciona um maior controlo sobre o fluxo das chamadas. A implementação descrita neste capítulo valida a escolha do *wrapper* AMI como a solução mais robusta para a invocação de aplicações *Dialplan*.

Capítulo 5

Suporte ASR e TTS

Este capítulo apresenta as abordagens estudadas e detalha a implementação das soluções escolhidas para fornecer suporte ASR e TTS ao `WMS Adapter`.

5.1 Estudo das Abordagens

Na solução atualmente adotada pela Altice Labs, o acesso a recursos de ASR e TTS pertencem a um módulo denominado WindlessMRCP. Este módulo está integrado ao WMS e é responsável por fornecer recursos de reconhecimento e síntese de voz. Esta abordagem implica que, e uma vez que os recursos de ASR e TTS são exclusivos ao WMS, qualquer aplicação que não utilize este componente fique privada do acesso a recursos de reconhecimento e síntese de voz. Com base neste paradigma, houve a necessidade de repensar a arquitetura utilizada para este tipo de sistemas, sendo que a solução adotada terá de consistir na integração dos recursos ASR e TTS num componente a montante do serviço, podendo este ser integrado no `WMS Adapter` ou no Asterisk. Esta solução terá de ser compatível com a versão atual do WMS e com eventuais serviços integrados no `WMS Adapter`.

Para integrar recursos ASR e TTS no sistema foi necessário analisar diversas abordagens de implementação. Este estudo permitiu compreender as vantagens e limitações de cada método, ajudando a escolher a solução mais adequada para as necessidades do projeto.

Nas secções abaixo serão apresentadas as soluções possíveis para solucionar o problema anteriormente apresentado.

5.1.1 Implementação do Módulo ARI

Como primeira abordagem, foi considerado a possibilidade de integrar estes recursos através da ARI. Por ser uma API já utilizada pelo `WMS Adapter`, foi ponderado adaptá-la para comunicar com serviços *cloud* de ASR e TTS.

Conforme apresentado na Figura 5.1, esta abordagem envolve a criação de um módulo que comunique diretamente com o *provider* de ASR e TTS. Dependendo da estratégia adotada no desenvolvimento deste módulo, podem ser atingidos tempos de resposta significativamente menores comparativamente com abordagens dependentes de intermediários. No entanto, devido à necessidade de desenvolvimento deste módulo de raiz, esta abordagem foi descartada a favor de soluções mais práticas.

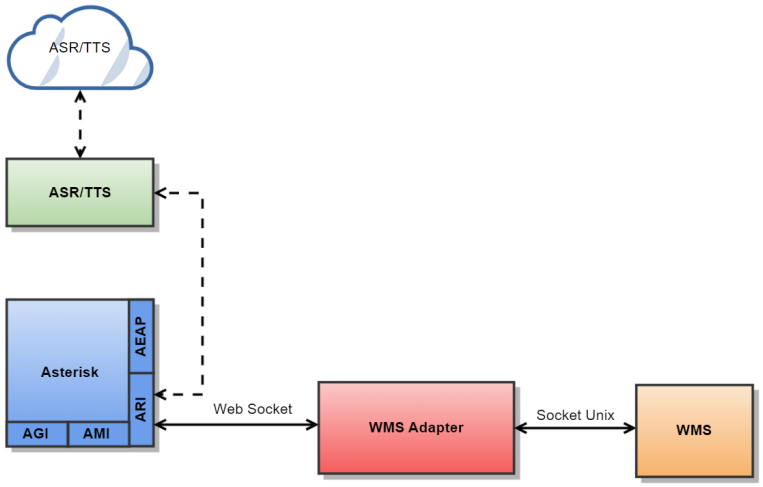


Figura 5.1: Implementação da ARI para recursos ASR e TTS

5.1.2 Implementação do Módulo AGI

Na procura de uma solução mais consolidada, foi ponderado a utilização da AGI. Semelhante à abordagem anteriormente mencionada, a integração realizada através desta API, representada na Figura 5.2, oferece uma implementação por intermédio de um módulo localizado entre o Asterisk e o *provider* de ASR e TTS. No entanto, trata-se de uma solução relativamente obsoleta devido ao desenvolvimento de interfaces dedicadas como a AEAP e o UniMRCP, projetadas tendo em conta este objetivo específico. Devido a estes fatores, a utilização da AGI foi descartada.

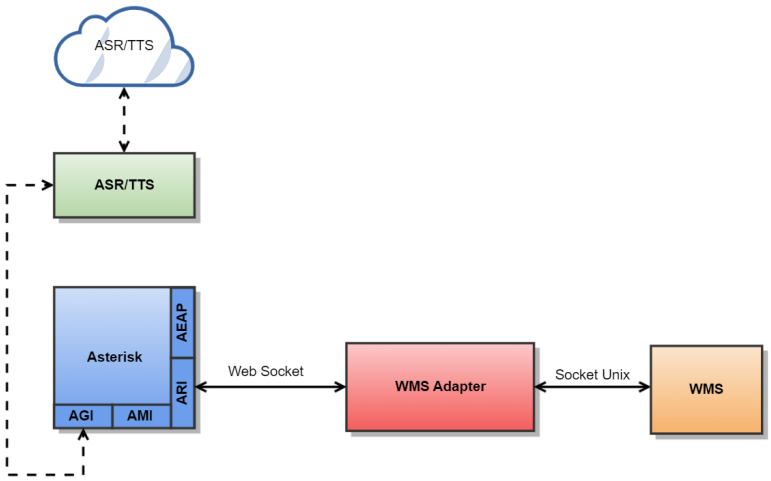


Figura 5.2: Implementação da AGI para recursos ASR e TTS

5.1.3 Implementação do Módulo AEAP

Criar um motor de reconhecimento de fala no Asterisk resulta num esforço significativo de implementação e recursos. Este exige a escrita de um módulo do Asterisk que implementa a API de *backend*. Com a utilização da AEAP, esta API é abstraída e a tradução pode ser efetuada externamente na linguagem de programação à escolha do utilizador.

Semelhante a outras soluções, a abordagem ilustrada na Figura 5.3 oferece uma integração simplificada. Contudo, a amostra fornecida pelo Asterisk não possui reconhecimento de início e fim de fala, o que se caracteriza num possível encargo adicional. Esta funcionalidade é particularmente importante, especialmente ao interagir com um *provider* onde a cobrança é feita por segundo, independentemente se a transmissão contém áudio ou apenas silêncio. Com a capacidade de deteção de início e fim de fala, é possível enviar apenas conteúdo útil, pagando apenas pelo tempo necessário.

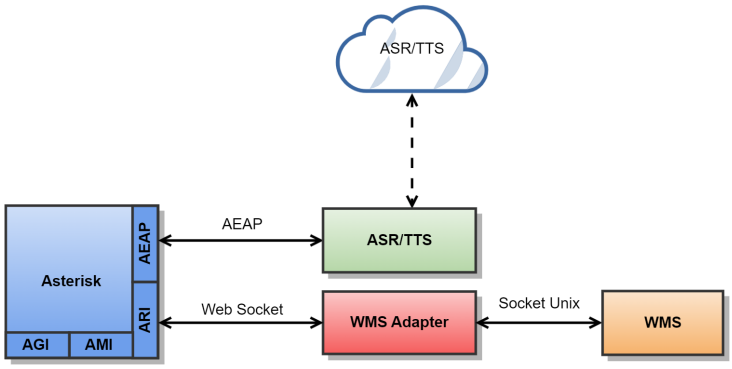


Figura 5.3: Implementação da AEAP para recursos ASR e TTS

5.1.4 Implementação do Módulo UniMRCP

O UniMRCP é uma implementação de código aberto que fornece uma implementação cliente-servidor MRCP, permitindo a comunicação entre servidores de fala e aplicações cliente [40]. A principal vantagem do UniMRCP é a sua capacidade de abstrair a complexidade da comunicação com serviços ASR e TTS, facilitando a integração de diferentes *providers* sem a necessidade de reescrever código. Apesar de ser uma abordagem que suporta funcionalidades de deteção de início e fim de fala, não permite uma comunicação direta com os servidores de fala, uma vez que é necessário um servidor UniMRCP atuar como intermediário, conforme representado na Figura 5.4. É importante salientar que o UniMRCP suporta oficialmente até à versão 19 do Asterisk, embora isso não exclua a possibilidade de suportar versões mais recentes.

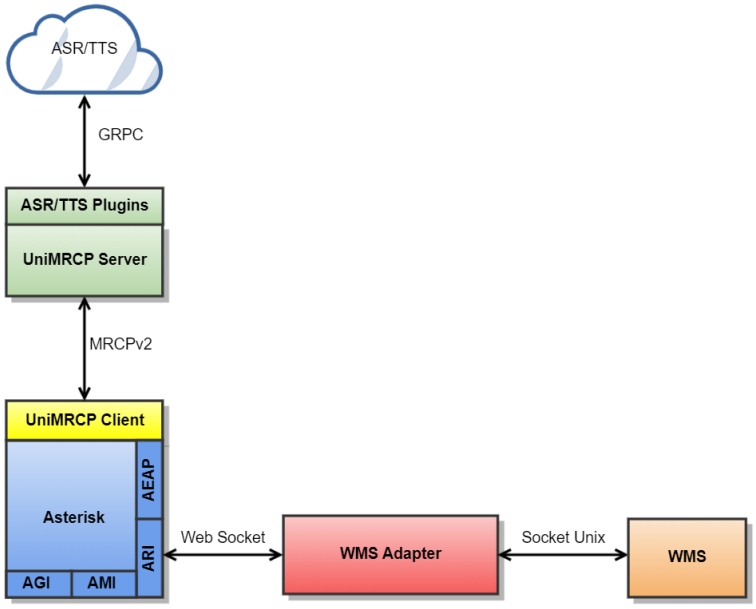


Figura 5.4: Implementação do UniMRCP para recursos ASR e TTS

5.1.5 Avaliação das Abordagens

Um resumo destas abordagens está representado na Tabela 5.1.

Requisitos/Pontos de avaliação	AGI	ARI	AEAP	UniMRCP Client
Suporta ASR?	⚠ Requer implementação	⚠ Requer implementação	✓	✓
Suporta TTS?	⚠ Requer implementação	⚠ Requer implementação	⚠ Requer implementação	✓
Suporta a versão mais recente do Asterisk?	✓	✓	✓	⚠ Suporta oficialmente até versão 19
Tem reconhecimento de início/fim de fala?	⚠ Requer implementação	⚠ Requer implementação	×	✓
É possível seguir uma abordagem de comunicação direta com o *provider*?	✓	✓	✓	× `UniMRCP Server`

Tabela 5.1: Comparação entre abordagens de integração ASR e TTS

Após uma análise detalhada das diferentes abordagens, foram consideradas duas soluções. Para a implementação conjunta de ASR e TTS, a abordagem UniMRCP revelou-se a mais adequada. Esta escolha deve-se à sua capacidade de interação com diversos *providers* e de suportar funcionalidades essenciais, como a deteção de início e fim de fala. Adicionalmente, devido à existência de um projeto base de implementação de recursos ASR desenvolvido pelo Asterisk, foi decidido utilizar também esta API, possibilitando uma comparação direta com o ASR do UniMRCP.

5.2 Implementação

Como referido anteriormente, foram adotadas duas abordagens no `WMS Adapter`: a abordagem AEAP e a abordagem UniMRCP. Após a implementação destas soluções, será possível compará-las tanto entre si como com a solução atual, levando em consideração critérios como responsividade, performance e determinismo. Esta análise permitirá identificar a solução que melhor atende às necessidades do `WMS Adapter`.

5.2.1 Módulo ASR via AEAP

A primeira implementação baseia-se no projeto de exemplo fornecido pelo Asterisk, que facilita a comunicação entre o Asterisk e o Google Speech Recognition [41]. Nesta implementação, é criado um *WebSocket* que fica à escuta de conexões provenientes do Asterisk. Uma vez que uma conexão é estabelecida, o áudio de fala pode ser enviado do Asterisk para o `WMS Adapter`, que, por sua vez, o encaminha para o *provider* de

reconhecimento de fala da Google. Ao receber um resultado de reconhecimento de fala com confiança suficiente, o `WMS Adapter` envia o resultado de volta ao Asterisk. Este reconhecimento continuará até que o cliente feche a conexão ou ocorra um erro.

O primeiro passo para utilizar esta abordagem será configurar o ficheiro de configuração "`aeap.conf`". Um exemplo de configuração está representado na Listagem 5.1.

```
[my-speech-to-text]
type=client
codecs=!all,ulaw
url=ws://127.0.0.1:9099
protocol=speech_to_text
```

Listagem 5.1: Configuração do ficheiro "`aeap.conf`"

Esta configuração define um motor de reconhecimento de fala e especifica que este atuará como cliente. É definido que irá utilizar apenas o *codec* "*ulaw*" e fornece o *Uniform Resource Locator* (URL) do servidor *WebSocket* ao qual o cliente irá se conectar. Por fim, especifica o protocolo `speech_to_text` para a comunicação. Esta configuração permite que o Asterisk utilize um serviço externo de reconhecimento de voz, conectando-se através de um cliente *WebSocket* quando a API de reconhecimento de fala do Asterisk é utilizada no *Dialplan*.

Para poder utilizar esta abordagem no Asterisk, resta configurar o *Dialplan* com um contexto que utiliza as funções da API de fala, representadas na Listagem 5.2.

```
[aeap]
exten => 550,1,NoOp()
        same => n,Answer()
        same => n,SpeechCreate(my-speech-to-text)
        same => n,SpeechBackground(hello-world)
        same => n,SpeechStart()
        same => n,Verbose(0,${SPEECH_TEXT(0)})
```

Listagem 5.2: Contexto do *Dialplan* para a abordagem AEAP

Neste contexto é definido uma extensão para realizar reconhecimento de fala usando a configuração `my-speech-to-text` realizada anteriormente. Quando o contexto é invocado, a chamada é atendida e é criado uma nova instância de reconhecimento de fala com a função `SpeechCreate()`. Enquanto o áudio do arquivo `hello-world` é reproduzido em segundo plano através da função com `Speech Background()`, o sistema escuta o que o chamador pronuncia através da função `SpeechStart()`. Por fim, o resultado do reconhecimento de fala é exibido na consola do Asterisk através do `Verbose()`, onde "`SPEECH_TEXT(0)`" contém o texto transcrito da fala reconhecida.

Após esta configuração, resta apenas iniciar o servidor *WebSocket* no porto 9099. Um resultado de um teste utilizando esta implementação pode ser observado na Figura 5.5.

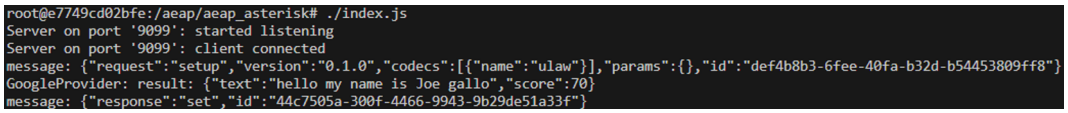


Figura 5.5: Resultado de um teste utilizando a implementação AEAP para recursos ASR

Neste exemplo foi recebido uma mensagem do *provider* com uma pontuação de 70. Esta pontuação significa que o Google Speech ASR tem 70% de confiança de que a transcrição do áudio está correta.

5.2.2 Módulos ASR e TTS via UniMRCP

Após a instalação bem-sucedida do UniMRCP, é necessário configurar o arquivo "`mrcp.conf`" para definir os perfis de ASR e TTS a utilizar, conforme apresentado na Listagem 5.3.

```
[general]
default-asr-profile = ums2
default-tts-profile = ums2
```

Listagem 5.3: Configuração ASR e TTS no ficheiro "`mrcp.conf`"

Em seguida, é necessário configurar os detalhes de versão, endereços IP e portas no arquivo "`mrcp.conf`" do perfil "`ums2`" definido como perfil de ASR e TTS, conforme ilustrado na Listagem 5.4.

```
[ums2]
version = 2

server-ip = 10.113.150.213
server-port = 8060

client-ip = 192.168.112.2
client-ext-ip = 10.113.137.250
client-port = 25097
sip-transport = udp

rtp-ip = 192.168.112.2
rtp-ext-ip = 10.113.137.250
rtp-port-min = 10000
rtp-port-max = 10010
```

Listagem 5.4: Configuração da versão e portos do UniMRCP no ficheiro "`mrcp.conf`"

Esta configuração especifica os detalhes de rede para a comunicação entre o cliente e o servidor MRCP, incluindo os endereços IP internos e externos, portas RTP, assim como a versão do protocolo.

Para integrar o UniMRCP com o Asterisk, será necessário configurar o *Dialplan* com dois contextos: um para ASR e outro para TTS. A Listagem 5.5 apresenta o contexto especifico para ASR.

```
[unimrcpasr]
exten => 550,1,NoOp()
        same => n,Answer()
        same => n,MRCPRecog(${Type}&f=hello-world)
        same => n,Verbose(1,${RECOGSTATUS},
        ${RECOG_COMPLETION_CAUSE},${RECOG_RESULT})
        same => n,Set(TYPE=ASR)
        same => n,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},
        ${INVITE_URI},${RECOGSTATUS},${RECOG_RESULT},
        ${RECOG_COMPLETION_CAUSE},${TYPE})
```

Listagem 5.5: Contexto do *Dialplan* para reconhecimento de fala da abordagem UniMRCP

Neste contexto, é atendida a chamada e iniciado o reconhecimento de fala através da função especifica do UniMRCP "`MRCPRecog()`". O resultado do reconhecimento é exibido na consola do Asterisk através da função `Verbose()`.

Se o reconhecimento for concluído, a variável `RECOGSTATUS` será definida como `OK`. Caso o reconhecimento não possa ser iniciado, a variável será definida como `ERROR`. Se a chamada for desligada enquanto a síntese ainda estiver em progresso, a variável será ajustada para `INTERRUPTED`. Já a variável `RECOG_COMPLETION_CAUSE` indica se o reconhecimento foi concluído com sucesso ou se ocorreu um erro, com valores possíveis de `"000"` para sucesso, `"001"` para sem correspondência e `"002"` caso não tenha recebido nenhuma entrada. Se o reconhecimento for concluído com êxito, a variável `RECOG_RESULT` será preenchida com um resultado detalhado recebido do servidor MRCP.

Para diferenciar os eventos de reconhecimento e síntese de fala ao reentrar na `Stasis`, é criada a variável `TYPE` e definida, neste caso, como "`ASR`". Por fim, a chamada volta ao `WMS Adapter` com os parâmetros que contêm os resultados, permitindo que esses valores sejam utilizados no `WMS Adapter`.

A Listagem 5.6 demonstra um teste bem sucedido para ASR utilizando este contexto.

```
 OK, 000, <?xml version="1.0"?>
<result>
  <interpretation grammar="builtin:speech/transcribe" confidence=
  "0.93">
    <instance>Olá o meu nome é Gonçalo</instance>
    <input mode="speech">Olá o meu nome é Gonçalo</input>
```

```
    </interpretation>
  </result>
```

Listagem 5.6: Resultado de um teste utilizando a implementação UniMRCP para recursos ASR

Neste exemplo, a variável `RECOG_STATUS` foi definida como `OK`, indicando que o reconhecimento foi concluído com sucesso. A variável `RECOG_COMPLETION_CAUSE` recebeu o valor `"000"`, confirmando o sucesso e o resultado do reconhecimento mostrou uma confiança de 93%.

O contexto relativo ao TTS do UniMRCP está representado na Listagem 5.7.

```
[unimrcptts]
exten => 550,1,NoOp()
        same => n,Answer()
        same => n,MRCPSynth(${MRCPSynth},l=${Language}
        &g=${Gender})
        same => n,Verbose(1,${SYNTHSTATUS})
        same => n,Set(TYPE=TTS)
        same => n,Stasis(WMSApp,${INVITE_TO},${INVITE_FROM},
        ${INVITE_URI},${SYNTHSTATUS},${TYPE})
```

Listagem 5.7: Contexto do *Dialplan* para síntese de fala da abordagem UniMRCP

Neste contexto, a chamada é atendida e iniciado a síntese de fala com a função `MRCPSynth()`, configurando parâmetros como idioma e género. O resultado da síntese é exibido na consola do Asterisk através da função `Verbose()` que contém a variável `SYNTHSTATUS`. Se a síntese for concluída com êxito, a variável será definida como `OK`. Caso ocorra algum erro, a variável será definida como `ERROR`. Se a chamada for desligada enquanto a síntese ainda estiver em progresso, a variável será definida como `INTERRUPTED`. Por fim, a variável `TYPE` é definida como "`TTS`" e a chamada reentra na `Stasis`.

5.3 Integração do WMS com o `WMS Adapter`

Nesta secção, abordaremos a integração dos recursos de ASR e TTS com o `WMS Adapter`. Esta integração permite que o `WMS Adapter` receba e processe eventos relacionados à síntese e reconhecimento de fala.

5.3.1 Integração TTS

Ao solicitar um recurso de TTS ao WMS, o evento `TTSPRO` é recebido pelo `WMS Adapter`. Para transportar os parâmetros relativos ao TTS e ASR pelo `WMS Adapter`, foram criadas as funções `GetTTS_ASR_Cap()` e `SetTTS_ASR_Cap()`. Após o evento

ATTCAP ser recebido, a função SetTTS_ASR_Cap() transmite os parâmetros necessários, enquanto a função GetTTS_ASR_Cap() os recupera posteriormente.

Ao receber o evento TTSPRO pelo WMS Adapter, é invocado a função TextToSpeech() com os parâmetros adequados para a síntese de fala. A Listagem 5.8 demonstra o código responsável por esta lógica.

```
language_tts, number_gender := _channel.GetTTS_ASR_Cap()

switch {
    case number_gender == 0:
        gender := "female"
        manager.TextToSpeech(cmd_ast.Al_char_val1, language_tts,
        gender, _channel)

    case number_gender == 1:
        gender := "male"
        manager.TextToSpeech(cmd_ast.Al_char_val1, language_tts,
        gender, _channel)
}
```

Listagem 5.8: Extrato de código ao receber o evento TTSPRO

Nesta função, as variáveis necessárias para a síntese de fala são substituídas no *Dialplan* através da função SetVariable(), incluindo parâmetros como a frase a ser transcrita, o idioma e o género. Por fim, o canal é enviado para o contexto relativo à síntese de fala. Na Listagem 5.9 está representado o trecho de código que realiza esta configuração.

```
name := "MRCPSynth"
err := _channel.SetVariable(name, sentence)

name2 := "Language"
err2 := _channel.SetVariable(name2, language_tts)

name3 := "Gender"
err3 := _channel.SetVariable(name3, gender)

context := "unimrcptts"
extension := "550"
priority := 1

_channel.Continue(context, extension, priority)
```

Listagem 5.9: Extrato de código da função TextToSpeech()

Após a execução da síntese, o canal ao reentrar na Stasis, verifica se a variável TYPE é igual a "TTS", retornando o evento TTS_RECEIVED da ARI. Neste evento é obtido o estado da síntese através da variável SYNTHSTATUS e apresentado o resultado no WMS Adapter. O excerto de código apresentado na Listagem 5.10 representa este processo.

```
_, _ = _channel.GetVariable("TYPE")

value := "SYNTHSTATUS"
variable, err := _channel.GetVariable(value)

ari_event_processor_logger.Info("UniMRCP TTS Completed:",
"Result:", variable)
```

Listagem 5.10: Extrato de código do evento TTS_RECEIVED

Um exemplo de uma síntese de fala executada com sucesso pode ser observado na Figura 5.6.

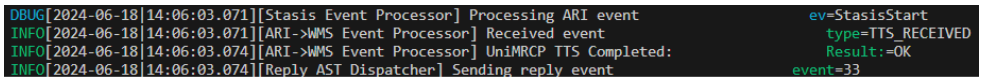


Figura 5.6: Resultado de uma síntese de voz no WMS Adapter

O resultado final é indicado pelo valor OK da variável SYNTHSTATUS recebida pela Stasis, o que confirma que a síntese foi realizada com sucesso.

5.3.2 Integração ASR

Ao contrário da implementação TTS, a integração de ASR no WMS Adapter foi projetada para utilizar duas abordagens distintas: AEAP e UniMRCP. A escolha da abordagem a utilizar é realizado pelo utilizador utilizando variáveis de ambiente.

A implementação inicia-se com a receção do evento STARTR. A Listagem 5.11 demonstra o código responsável por esta lógica inicial.

```
language_asr, _ := _channel.GetTTS_ASR_Cap()

manager.SpeechToText(language_asr, _channel)
```

Listagem 5.11: Extrato de código ao receber o evento STARTR

Ao receber este evento, a função SpeechToText() é invocada com os parâmetros adequados para a síntese de fala. De forma similar à implementação TTS, o WMS Adapter recolhe os parâmetros necessários para a execução do reconhecimento de fala utilizando a função GetTTS_ASR_Cap(). A Listagem 5.12 demonstra o código responsável por esta lógica.

```
name := "Type"
err := _channel.SetVariable(name, language_asr)

switch cl.asrType {
    case "unimrcpasr":
        context = "unimrcpasr"
    case "aeap":
```

```
            context = "aeap"
        }

    extension = "550"
    priority = 1

    _channel.Continue(context, extension, priority)
```

Listagem 5.12: Extrato de código da função `SpeechToText()`

Primeiramente, a variável é definida com o valor apropriado para o idioma do ASR. Em seguida, dependendo da abordagem escolhida, o `WMS Adapter` distingue qual a abordagem ASR será utilizada, enviando o canal para o contexto correspondente para a execução do reconhecimento de fala.

Após a execução do reconhecimento de fala, o canal retorna à `Stasis`, onde é verificado se a variável `TYPE` corresponde a `"ASR"`. Neste ponto, o evento `ASR_RECEIVED` é retornado. A Listagem 5.13 apresenta o código responsável por este processo.

```
switch asrProvider {
  case "unimrcpasr":
        value := "RECOG_RESULT"
        result_unimrcp, err := _channel.GetVariable(value)

        ari_event_processor_logger.Info("UniMRCP ASR Completed:",
        "Result:\n", result_unimrcp)

    case "aeap":
        value := "SPEECH_TEXT(0)"
        result_aeap, err := _channel.GetVariable(value)

        ari_event_processor_logger.Info("AEAP ASR Completed:",
        "Result:\n", result_aeap)
}
```

Listagem 5.13: Extrato de código do evento `ASR_RECEIVED`

Para a abordagem UniMRCP, a variável `RECOG_RESULT` é utilizada para obter o resultado do reconhecimento, enquanto que para a AEAP, a variável `SPEECH_TEXT(0)` é utilizada. Estes resultados são então apresentados no `WMS Adapter`, proporcionando informações sobre o desempenho do reconhecimento de fala. Um exemplo de um reconhecimento de fala executado com sucesso pode ser observado na Figura 5.7.

```
DBUG[2024-06-18|14:03:01.618][Stasis Event Processor] Processing ARI event                ev=StasisStart
INFO[2024-06-18|14:03:01.618][ARI->WMS Event Processor] Received event                    type=ASR_RECEIVED
INFO[2024-06-18|14:03:01.620][ARI->WMS Event Processor] UniMRCP ASR Completed:            Result:
-"<?xml version=\"1.0\"?>\n<result>\n  <interpretation grammar=\"builtin:speech/transcribe\" confidence=\"0.93\">\n    <instance>Olá o meu nome é
 Gonçalo</instance>\n    <input mode=\"speech\">Olá o meu nome é Gonçalo</input>\n  </interpretation>\n</result>"
INFO[2024-06-18|14:03:01.621][ARI->WMS Event Processor] Recognition:                      Result:=OK
INFO[2024-06-18|14:03:01.623][ARI->WMS Event Processor] 000-Success, 001-NoMatch, 002-NoInput:  Result:=000
INFO[2024-06-18|14:03:01.623][Reply AST Dispatcher] Sending reply event                   event=31
```

Figura 5.7: Resultado de um reconhecimento de voz no `WMS Adapter`

O resultado final é apresentado na consola, com os valores "OK" e "000" indicando que o reconhecimento de fala foi bem-sucedido e que não ocorreram erros durante o processo.

5.3.3 Integração com o WMS

Para utilizar o UniMRCP com o WMS-CMAN foi necessário modificar e adicionar novas funções, de modo a enviar os eventos corretos e reproduzir o comportamento da abordagem já existente do WindlessMRCP para todos os comandos InoAPI. O objetivo era permitir que o utilizador escolhesse, através de variáveis de ambiente, entre as duas abordagens para utilizar ASR e TTS. Desta forma, foi preciso separá-las no código-fonte do WMS-CMAN, desenvolvido em linguagem C, para que ambos funcionassem de maneira independente.

Quando o utilizador inicia o serviço WMS Adapter, diferentes comandos são processados pelo WMS. Cada comando é então associado a uma função específica, conforme definido nos diversos *case statements*. Dependendo do *provider*, diferentes funções são invocadas para realizar ASR e TTS.

Alterações TTS

No caso do TTS, a função TTSProcess() foi modificada para verificar o *provider* configurado. Caso o comando TTSPRO seja solicitado, a função é invocada, processando os comandos de síntese de fala tal como demonstra a Listagem 5.14.

```
case TTSPRO:
  if ((error = TTSProcess(chinfo, cmd)) != VSRV_NOERROR)
     trsERROR("ProcessCmd: TTSProcess: return error %d", error);
  break;
```

Listagem 5.14: Extrato de código do *case* TTSPRO

Dentro da função TTSProcess(), é verificado o valor da variável ttsProvider. Caso seja um, irá ser adotada a solução relativa ao UniMRCP. Esta implementação está representado na Listagem 5.15.

```
if (ttsProvider==1) {
    cmdast.cmd_id = TTSPRO;
    cmdast.al_int_val1 = 1;
    strncpy(cmdast.al_char_val1, cmd->tts_rattrib.source,
    sizeof(cmdast.al_char_val1));

    if (asteriskSendCmd(idx, &cmdast) == 0){
        (...)
    }
}
```

Listagem 5.15: Extrato de código da função TTSProcess()

Nesta solução, é enviado para o Asterisk um comando que contém informações como idioma e género a utilizar no TTS, indicado pelo utilizador através de comandos `InoAPI`. Se o envio for bem-sucedido, o processo continua normalmente. Caso contrário, um erro é registado e uma resposta de erro é enviada.

Alterações ASR

No caso do ASR, foi necessário criar uma nova função responsável pelo reconhecimento de voz para lidar com diferentes *providers* denominada `StartAsteriskRecog()`. Esta distinção está representada na Listagem 5.16.

```
case STARTR:
  if(asrProvider==3) {
    if ((error = StartAsteriskRecog(chinfo, cmd)) != VSRV_NOERROR)
        (...)
  } else {
      if(asrMrcpClient==0){
        if ((error = StartRecog(chinfo, cmd)) != VSRV_NOERROR)
            (...)
      }
    }
break;
```

Listagem 5.16: Extrato de código do *case* `STARTR`

Do mesmo modo que acontece para o TTS, a função `StartAsteriskRecog()` é responsável por enviar os comandos apropriados para o Asterisk para a implementação UniMRCP. Esta implementação está representada na Listagem 5.17.

```
int StartAsteriskRecog(CHINFO *chinfo, cmd_format *cmd) {
   cmdast.cmd_id = STARTR;
   cmdast.al_int_val1 = 1;

    if ((error = asteriskSendCmd(idx, &cmdast)) == -1) {
        (...)
    }
}
```

Listagem 5.17: Extrato de código da função `StartAsteriskRecog()`

A função `StartAsteriskRecog()` envia os comandos ao Asterisk para iniciar o reconhecimento de voz, configurando os parâmetros necessários conforme as especificações do comando recebido. Se o envio falhar, um erro é registado e o processo é interrompido. Caso contrário, a operação continua normalmente.

5.3.4 Integração com Aplicações em Go

De modo semelhante aos requisitos iniciais do `WMS Adapter`, foi necessário realizar a integração de ASR e TTS com aplicações em `Go`. Esta integração é iniciada pelo

utilizador através de uma chamada para o Asterisk, que serve como ponto de partida para todos os processos subsequentes.

Ao realizar uma chamada, o evento é capturado pela ARI e o utilizador é então convidado a escolher entre duas opções: síntese ou reconhecimento de fala. Para a opção de síntese de fala, o utilizador deve fornecer a frase a ser sintetizada, juntamente com o idioma e o género desejado. Estes detalhes são importantes para comprovar o bom funcionamento de todos as características da síntese de fala. O serviço chama a função responsável por realizar TTS, que utiliza o UniMRCP para realizar a síntese de fala.

No caso do reconhecimento de fala, o utilizador seleciona o idioma para a transcrição e define o tempo limite de entrada. O reconhecimento de fala é então realizado através da chamada à função responsável por realizar ASR, que novamente utiliza o UniMRCP para processar a entrada de voz e retornar a transcrição ao canal.

Após a execução das operações de TTS ou ASR, os resultados são capturados e apresentados ao utilizador.

5.4 Comparação de Resultados

Para uma melhor compreensão do desempenho das abordagens UniMRCP e WindlessMRCP, foram realizados cerca de cinquenta testes de síntese de fala para cada uma. Uma vez que o AEAP não possui funcionalidades de TTS, não foi feita qualquer análise para este sistema. Os testes foram conduzidos com dois tamanhos diferentes de texto: o texto grande, com 173 caracteres, e o pequeno, com 23 caracteres. Os resultados obtidos estão representados na Tabela 5.2.

Comprimento do Texto	**Sistema**	**Tempo Médio de Processamento (s)**	**Diferença Média de Tempo (s)**
23 Caracteres	WindlessMRCP	1,526	0,236
	UniMRCP	0,354	0,095
173 Caracteres	WindlessMRCP	1,701	0,399
	UniMRCP	0,688	0,265

Tabela 5.2: Comparação dos resultados entre WindlessMRCP e UniMRCP

O tempo médio de processamento para o texto pequeno com o WindlessMRCP foi de aproximadamente 1,526 segundos, enquanto com o UniMRCP foi de 0,354 segundos. Para o texto grande, o WindlessMRCP demorou em média 1,701 segundos, em contraste com os 0,688 segundos do UniMRCP. Estes resultados indicam uma vantagem significativa do UniMRCP em ambos os casos, apresentando tempos de resposta consideravelmente menores.

Além do desempenho geral, a consistência dos resultados também foi um fator relevante. A diferença média de tempos evidencia ainda mais a superioridade do

UniMRCP. Para textos pequenos, a diferença média foi de 0,236 segundos para o WindlessMRCP e 0,095 segundos para o UniMRCP. No caso dos textos grandes, a diferença foi de 0,399 segundos para o WindlessMRCP e 0,265 segundos para o UniMRCP. Estes valores confirmam que, além de ser mais rápido, o UniMRCP é também mais estável.

Os testes realizados demonstram que o UniMRCP supera significativamente o WindlessMRCP tanto para textos pequenos quanto grandes. Esta superioridade reflete-se não só em tempos de resposta menores, mas também em maior consistência e estabilidade dos resultados.

Capítulo 6

Conclusão e Trabalho Futuro

Este trabalho teve como objetivo evoluir a aplicação `WMS Adapter` com funcionalidades avançadas de controlo de chamadas, integrando recursos de ASR e TTS, além de possibilitar a sua utilização em ambientes conteinerizados. A evolução tecnológica do Asterisk motivou a necessidade de aprimorar a sua integração com o WMS, utilizando o `WMS Adapter` para aproveitar as APIs disponibilizadas pelo Asterisk e interligar estes dois sistemas.

A implementação de chamadas de saída foi um dos primeiros objetivos estabelecidos, permitindo a gestão eficiente destas chamadas utilizando a aplicação `Stasis` do Asterisk. A funcionalidade de realização de *route endpoints* foi também integrada, garantindo uma flexibilidade adicional na gestão das comunicações.

A integração de recursos ASR e TTS foi uma das etapas mais desafiantes e cruciais do projeto, onde várias abordagens foram exploradas para a implementação destes recursos. Como implementação final, optou-se pelo uso das abordagens AEAP para ASR, e UniMRCP para ASR e TTS.

A adaptação do `WMS Adapter` para ambientes conteinerizados foi outra área de interesse, utilizando tecnologias como o Docker para criar um ambiente escalável. A conteinerização do Asterisk e do `WMS Adapter` permitiu uma maior flexibilidade na implementação do sistema, facilitando o *deployment* para diferentes ambientes.

No geral, o projeto foi concluído e todos os objetivos inicialmente propostos foram alcançados.

Apesar dos resultados obtidos serem promissores, existem melhorias que podem ser implementadas. Em primeiro lugar, será essencial continuar a evolução do `WMS`

`Adapter`. Deste modo, será possível tornar o `WMS Adapter` cada vez mais completo, para no futuro eliminar qualquer dependência do WMS. Este tornar-se-á numa aplicação `Stasis` totalmente independente, conforme demonstra a Figura 6.1.

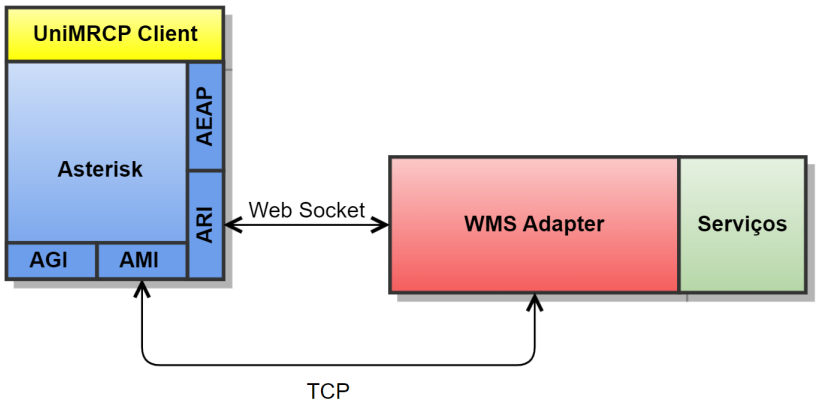


Figura 6.1: Futura versão do `WMS Adapter`

Além disso, será necessário explorar a integração deste sistema em ambientes Kubernetes, assim como a realização de testes de carga para garantir a robustez da aplicação em ambientes de produção.

Referências

[1] Asterisk, "Getting started with asterisk." Available at https://www.asterisk.org/get-started/. (Último acesso: 11-01-2024). [Citado nas páginas 1 e 5]

[2] Altice, "Inovação | altice labs portugal." Available at https://www.altice.pt/pt/inovacao/inovacao-para-liderar. (Último acesso: 28-01-2024). [Citado na página 2]

[3] Altice, "Where we are - altice labs." Available at https://www.alticelabs.com/contacts/. (Último acesso: 28-01-2024). [Citado na página 3]

[4] Asterisk, "Asterisk." Available at https://www.asterisk.org/. (Último acesso: 11-01-2024). [Citado na página 5]

[5] J. Meggelen, R. Bryant, and L. Madsen, *Asterisk: The Definitive Guide*. O'Reilly Media, Inc., 5th ed., 2019. [Citado nas páginas 5 e 52]

[6] Asterisk, "Ivr." Available at https://www.asterisk.org/get-started/applications/ivr/. (Último acesso: 18-01-2024). [Citado na página 5]

[7] Asterisk, "Asterisk architecture." Available at https://docs.asterisk.org/Fundamentals/Asterisk-Architecture/Asterisk-Architecture-The-Big-Picture/. (Último acesso: 11-01-2024). [Citado nas páginas ix, 6 e 7]

[8] Asterisk, "The asterisk dialplan." Available at https://docs.asterisk.org/Configuration/Dialplan/. (Último acesso: 17-01-2024). [Citado na página 8]

[9] Asterisk, "Contexts, extensions, and priorities." Available at https://docs.asterisk.org/Configuration/Dialplan/Contexts-Extensions-and-Priorities/. (Último acesso: 17-01-2024). [Citado na página 8]

[10] Asterisk, "Dialplan applications." Available at https://docs.asterisk.org/Asterisk_21_Documentation/API_Documentation/Dialplan_Applications/. (Último acesso: 17-01-2024). [Citado na página 9]

[11] Asterisk, "Stasis dialplan application." Available at https://docs.asterisk.org/Asterisk_21_Documentation/API_Documentation/Dialplan_

Applications/Stasis/. (Último acesso: 17-01-2024). [Citado na página 9]

[12] Asterisk, "The evolution of asterisk apis." Available at https://docs.asterisk.org/Configuration/Interfaces/Asterisk-REST-Interface-ARI/. (Último acesso: 11-01-2024). [Citado nas páginas ix, 10 e 11]

[13] Asterisk, "The asterisk manager tcp ip api." Available at https://docs.asterisk.org/Configuration/Interfaces/Asterisk-Manager-Interface-AMI/The-Asterisk-Manager-TCP-IP-API/. (Último acesso: 16-01-2024). [Citado na página 10]

[14] J. C. Colp, "What really is ari?." Available at https://www.asterisk.org/what-really-is-ari/. (Último acesso: 15-01-2024). [Citado na página 10]

[15] K. Harwell, "Asterisk external application protocol: An intro." Available at https://www.asterisk.org/asterisk-external-application-protocol-an-intro/. (Último acesso: 08-04-2024). [Citado na página 12]

[16] Altice, "Windless media server - descrição técnica de alto nível." Altice Labs (Confidencial). [Citado nas páginas ix e 13]

[17] F. Delfim, "Windless media server - arquitetura detalhada das versões 3.x." Altice Labs (Confidencial). [Citado nas páginas ix e 14]

[18] P. N. Pinto, "Windless media server - cman." Altice Labs (Confidencial). [Citado nas páginas ix, 15 e 16]

[19] E. Schooler, J. Rosenberg, H. Schulzrinne, A. Johnston, G. Camarillo, J. Peterson, R. Sparks, and M. J. Handley, "SIP: Session Initiation Protocol." RFC 3261, July 2002. [Citado nas páginas 17 e 18]

[20] B. T. C. Jorge, "Segurança e privacidade numa infraestrutura de voip," mestrado integrado em engenharia eletrotécnica e de computadores, Faculdade de Engenharia da Universidade do Porto, 2017. Available at https://repositorio-aberto.up.pt/bitstream/10216/102609/2/180841.pdf. [Citado nas páginas ix e 18]

[21] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, *SIP: session initiation protocol*. Network Working Group, 1999. [Citado na página 19]

[22] A. B. Johnston, *SIP: understanding the session initiation protocol*. Artech House, 4th ed., 2015. [Citado nas páginas ix e 20]

[23] C. Perkins, M. J. Handley, and V. Jacobson, "SDP: Session Description Protocol." RFC 4566, July 2006. [Citado nas páginas xiii, 20 e 22]

[24] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications." RFC 3550, July 2003. [Citado nas páginas 22 e 23]

[25] M. M. Alani, *Guide to OSI and TCP/IP Models*. Springer Cham, 2014. [Citado na página 22]

[26] D. Irwin and J. Slay, "Extracting evidence related to voip calls," in *Advances in Digital Forensics VII, IFIP AICT 361*, pp. 221–228, 01 2011. [Citado nas páginas xi e 22]

[27] Teluu, "About pjsip." Available at `https://www.pjsip.org/about.htm`. (Último acesso: 02-05-2024). [Citado na página 24]

[28] Teluu, "Pjsip overview." Available at `https://docs.pjsip.org/en/latest/overview/intro.html`. (Último acesso: 02-05-2024). [Citado nas páginas ix e 24]

[29] B. Eberman, P. Monaco, and S. Shanmugham, "A Media Resource Control Protocol (MRCP) Developed by Cisco, Nuance, and Speechworks." RFC 4463, Apr. 2006. [Citado na página 25]

[30] D. Burke, *Speech processing for ip networks: Media resource control protocol (MRCP)*. John Wiley & Sons, 2007. [Citado nas páginas ix, 25, 26, 27, 28 e 29]

[31] S. D. Burnett, *Media Resource Control Protocol Version 2 (MRCPv2) draft-ietf-speechsc-mrcpv2-28*. Wiley, 2012. [Citado na página 25]

[32] J. Lai, C.-M. Karat, and N. Yankelovich, *The Human-Computer Interaction Handbook*, vol. 1. CRC Press, 2007. [Citado na página 30]

[33] D. Yu and L. Deng, *Automatic Speech Recognition: A Deep Learning Approach*, vol. 1 of *Signals and Communication Technology*. Springer London, 2015. [Citado nas páginas ix e 31]

[34] A. Balyan, S. S. Agrawal, and A. Dev, "Speech synthesis: a review," *International Journal of Engineering Research & Technology (IJERT)*, vol. 2, no. 6, pp. 57–75, 2013. [Citado na página 32]

[35] A. Trivedi, N. Pant, P. Shah, S. Sonik, and S. Agrawal, "Speech to text and text to speech recognition systems-areview," *IOSR J. Comput. Eng*, vol. 20, no. 2, pp. 36–43, 2018. [Citado nas páginas ix e 32]

[36] H. Sommerfeldt, "Free voip softphone for windows." Available at `https://lite.phoner.de/index_en.htm`. (Último acesso: 22-04-2024). [Citado na página 38]

[37] CyCoreSystems, "Agi wrapper." Available at `https://github.com/CyCoreSystems/agi`. (Último acesso: 9-06-2024). [Citado na página 49]

[38] H. Marques, "Ami wrapper." Available at `https://github.com/heltonmarx/goami`. (Último acesso: 27-02-2024). [Citado na página 50]

[39] Asterisk, "Exec()." Available at `https://docs.asterisk.org/Latest_API/API_Documentation/Dialplan_Applications/Exec/`. (Último acesso: 10-06-2024). [Citado na página 53]

[40] A. Chaloyan, "Uni mrcp." Available at `https://www.unimrcp.org/faq`. (Último acesso: 01-03-2024). [Citado na página 58]

[41] Asterisk, "Asterisk external speech to text application." Available at `https://github.com/asterisk/aeap-speech-to-text`. (Último acesso: 14-06-2024). [Citado na página 59]

[42] Docker, "Docker overview." Available at `https://docs.docker.com/get-started/overview/`. (Último acesso: 02-06-2024). [Citado na página 77]

[43] Docker, "What is a container?." Available at `https://docs.docker.com/guides/docker-concepts/the-basics/what-is-a-container/`. (Último acesso: 02-06-2024). [Citado na página 77]

[44] Docker, "Dockerfile reference." Available at `https://docs.docker.com/reference/dockerfile/`. (Último acesso: 02-06-2024). [Citado na página 77]

[45] Docker, "Multi-stage builds." Available at `https://docs.docker.com/build/building/multi-stage/`. (Último acesso: 02-06-2024). [Citado na página 78]

[46] Docker, "Docker compose overview." Available at `https://docs.docker.com/compose/`. (Último acesso: 02-06-2024). [Citado na página 79]

Anexo A

Processo de Conteinerização do Asterisk e WMS Adapter

Este anexo descreve o processo de conteinerização do Asterisk e `WMS Adapter` utilizando o Docker. Este processo permite maior flexibilidade e portabilidade, facilitando a gestão e a escalabilidade dos serviços. O processo está dividido em duas partes: a construção da imagem Docker do Asterisk e a configuração do Docker Compose para orquestrar os *containers* das imagens relativas ao Asterisk e WMS-CMAN.

A.1 Asterisk em Docker

Para colocar o Asterisk em ambiente de virtualização, foi utilizado o Docker. O Docker é uma plataforma aberta que permite a construção, envio e execução de aplicações de forma isolada em *containers* [42]. Os *containers* são ambientes leves e portáteis que garantem que uma aplicação funcione de maneira consistente, independentemente do ambiente onde é executada, seja no desenvolvimento, teste ou produção [43].

Para colocar o Asterisk num *container* Docker, foi necessário criar uma imagem do mesmo através de um Dockerfile. Um Dockerfile é um *script* de configuração automatizado que contém um conjunto de instruções para criar uma imagem Docker [44]. Estas instruções especificam os comandos a serem executados para configurar o ambiente dentro do *container*, incluindo a instalação de pacotes, a configuração de

variáveis de ambiente e a cópia de arquivos necessários. A criação da imagem a partir do Dockerfile garante que o ambiente de execução do Asterisk seja reproduzível e consistente em qualquer sistema que suporte Docker.

O Dockerfile utilizado para a construção da imagem Docker do Asterisk está dividido em duas partes, utilizando uma abordagem de construção *multistage*. Esta técnica é utilizada para otimizar o processo de construção, separando a etapa de compilação e configuração, da etapa de criação da imagem final. A utilização do *multistage* é vantajosa devido à possibilidade de criação de imagens Docker mais leves e eficientes. Durante o estágio de compilação, todas as ferramentas e dependências necessárias para construir o Asterisk são incluídas, mas na imagem final, apenas os artefactos essenciais são copiados, descartando qualquer sobrecarga desnecessária. Isso resulta numa imagem mais compacta, que ocupa menos espaço e melhora o tempo de *download* e *deploy* [45].

A.1.1 *Build Stage*

Na primeira parte da construção *multi-stage*, denominada *"build-stage"*, foram realizadas diversas etapas para configurar e construir a imagem Docker. Como imagem base do *container*, foi escolhido o Ubuntu 20.04 para fornecer um ambiente Linux estável. Para garantir o correto funcionamento do Asterisk, foram instaladas todas as dependências necessárias para a sua compilação, incluindo bibliotecas e ferramentas essenciais.

Posteriormente, foi realizado o processo de configuração do Asterisk a partir do repositório oficial no GitHub, incluindo a compilação de binários, instalação, assim como a criação de exemplos de configuração. Para integrar o UniMRCP no *container*, foram instaladas as suas dependências e feita toda a configuração a partir do repositório oficial no GitHub. A extensão do Asterisk fornecida pelo UniMRCP também foi instalada para permitir a integração completa.

A.1.2 *Runtime Stage*

Na segunda parte da construção *multi-stage*, denominada *"runtime-stage"*, foram realizados vários passos para configurar e otimizar a imagem final do Docker. Primeiramente, foram instaladas as dependências específicas do UniMRCP e AEAP. Em seguida, foram copiados para a imagem final os arquivos e diretórios do Asterisk gerados na fase anterior, incluindo as configurações, binários e módulos essenciais. Foram também copiados arquivos e diretórios do UniMRCP e AEAP necessários para o seu funcionamento com o Asterisk.

Além disso, foi criado e copiado um *script* denominado `docker-entrypoint.sh` para o *container*. Este *script* configura e inicializa o Asterisk dentro do *container* Docker, ajustando as definições com base nas variáveis de ambiente fornecidas.

Adicionalmente, copia os arquivos de configuração personalizados para o diretório responsável por armazenar os ficheiros de configuração do Asterisk, uma vez que não é possível configurá-los diretamente através de variáveis de ambiente.

Quanto ao UniMRCP, o *script* modifica o arquivo `mrcp.conf`, ajustando as configurações do servidor e do cliente MRCP. Finalmente, o Asterisk é inicializado.

A.2 Docker Compose

Para orquestrar os *containeres* Docker do Asterisk e do WMS-CMAN, foi utilizado o Docker Compose. O Docker Compose é uma ferramenta que permite definir e gerir aplicações Docker *multi-containeres*. Utilizando um único arquivo de configuração `YAML`, é possível configurar todos os serviços necessários e as interações entre eles [46].

No arquivo `docker-compose.yml`, são definidos dois serviços principais: o serviço Asterisk e o serviço WMS-CMAN. Ambos os serviços utilizam imagens Docker provenientes do GitHub Actions, garantindo que a construção e a publicação das imagens seja automatizada sempre que um *commit* é executado no repositório do projeto no GitHub.

A.2.1 Serviço Asterisk

Para este serviço são montados alguns volumes, nomeadamente os arquivos de configuração mencionados acima, arquivos de áudio e credenciais necessárias para a integração com o AEAP. Além disso, são mapeadas portas específicas para permitir a comunicação externa com o Asterisk, incluindo portas para SIP, RTP, ARI e AMI. Como ficheiro de variáveis de ambiente, utiliza o ficheiro `env_asterisk`. Este ficheiro define as configurações para o funcionamento do Asterisk no *conteiner* Docker, permitindo a sua configuração flexível sem a necessidade de modificar diretamente os arquivos de configuração. Este arquivo especifica o endereço IP externo utilizado do Asterisk e ativa a ARI, juntamente com o nome de utilizador, senha, porta e nome do servidor. Além disso, configura a AMI especificando a porta, endereço de ligação e utilizadores permitidos com as suas respetivas permissões de leitura e escrita. As variáveis incluem também a definição das portas RTP e configurações para o UniMRCP, especificando o servidor e porta do MRCP.

A.2.2 Serviço WMS-CMAN

O funcionamento do serviço WMS-CMAN está intimamente ligado ao serviço Asterisk. Sem o Asterisk em execução, este serviço não pode ser iniciado. Este vínculo implica que a disponibilidade do WMS-CMAN dependa diretamente da operação adequada do Asterisk. As configurações de ambiente deste serviço são fornecidas

pelo arquivo `env_wms_cman`, que abrange uma variedade de configurações essenciais incluindo licenças, registo de *logs*, configurações SIP, configurações de comunicação, configurações de *Resource Manager*, ASR, TTS e detalhes específicos do SIP *Stack*.

Esta metodologia de conteinerização contribui significativamente para a modernização da infraestrutura, garantindo que as aplicações possam ser facilmente desenvolvidas, testadas e implementadas de maneira uniforme e controlada.