UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ESTILLAC LINS MACIEL BORGES FILHO

UMA FERRAMENTA PARA PROJETO DE SISTEMAS DE DIÁLOGOS PARA CALL CENTER BASEADOS EM ASTERISK

DM: 30 / 2013

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém – Pará – Brasil
2013

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ESTILLAC LINS MACIEL BORGES FILHO

UMA FERRAMENTA PARA PROJETO DE SISTEMAS DE DIÁLOGOS PARA CALL CENTER BASEADOS EM ASTERISK

Dissertação submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da UFPA para a obtenção do Grau de Mestre em Engenharia Elétrica na área de Computação Aplicada.

Orientador: Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém – Pará – Brasil
2013

Dados Internacionais de Catalogação-na-Publicação (CIP)

Borges Filho, Estillac Lins Maciel, 1983-
Uma ferramenta para projeto de sistemas de diálogos para call center baseados em asterisk / Estillac Lins Maciel Borges Filho. - 2013.

Orientador: Aldebaro Barreto da Rocha Klautau Júnior.
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2013.

1. Asterisk (programa de computador). 2. Reconhecimento automático da voz. 3. Clientes - contato. I. Título.

CDD 22. ed. 005.3

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

"UMA FERRAMENTA PARA PROJETO DE SISTEMAS DE DIÁLOGOS PARA CALL CENTER BASEADOS EM ASTERISK"

AUTOR: **ESTILLAC LINS MACIEL BORGES FILHO**

DISSERTAÇÃO DE MESTRADO SUBMETIDA À BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SENDO JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE COMPUTAÇÃO APLICADA.

APROVADA EM: 16 / 12 / 2013.

BANCA EXAMINADORA:

Prof. Dr. Aldebaro Barreto da Rocha Klautau Junior
(Orientador – PPGEE/UFPA)

Prof. Dr. Lamartine Vilar de Souza
(Avaliador Externo – CAAB/UFPA)

Prof. Dr. Nelson Cruz Sampaio Neto
(Avaliador Externo – ICEN/UFPA)

VISTO:

Prof. Dr. Evaldo Gonçalves Pelaes
(Coordenador do PPGEE/ITEC/UFPA)

Dedico este trabalho à minha esposa Maria Adrina, que nunca me deixou desistir de meus sonhos.

AGRADECIMENTOS

Foram seis anos e meio dedicados à obtenção do título de mestre, incluindo dois programas de pós-graduação diferentes; foram muitos os obstáculos que surgiram neste período e tiveram que ser superados; foram vários os momentos em que eu pensei em desistir, mas segui em frente e aqui está minha dissertação. Não há como não agradecer a Deus e a Nossa Senhora de Nazaré por mais essa graça alcançada e também à minha esposa, Maria Adrina, que jamais me deixou desanimar e que segurou as pontas naqueles momentos em que precisei ficar ausente para me dedicar a este trabalho.

Agradeço também aos meus dois anjos, meus filhos Bernardo e Estrela Maria, por tornarem tudo mais fácil com seus sorrisos, carinhos e alegria de viver.

Agradeço aos meus pais, Estillac e Bernadete, por me darem tudo aquilo que um filho necessita: amor, família e educação. O resto se conquista.

Agradeço aos meus irmãos, José e João Luiz, pelo amor fraterno, sempre presente, mesmo nos momentos distantes. Ao José, em especial, agradeço ter me recomendado ao Prof. Aldebaro quando decidi recomeçar meu mestrado em Belém.

Agradeço a todos meus inúmeros familiares e amigos pelo incentivo, consciente ou inconsciente, que sempre me deram em cada etapa de minha vida. Gostaria muito de citar todos, mas o espaço é curto e espero que cada um se sinta lembrado.

Agradeço ao Prof. Carlos Henrique, que primeiro me orientou nesta minha caminhada, pelos ensinamentos e amizade, assim como ao Prof. Aldebaro Klautau, meu orientador, por confiar no meu trabalho e pelo constante incentivo.

Agradeço a todos do LaPS por me acolherem tão bem durante o mestrado, em especial ao Pedro, pela inestimável ajuda ao longo de meu trabalho; à Kelly, pelo apoio prestado em diversos momentos; e ao Maurício, pelas boas sugestões.

Agradeço à UFPA, através do PPGEE, a oportunidade de realizar esse curso; e aos Professores Lamartine e Nelson, por participarem de minha banca avaliadora.

Em suma, a todos que, de alguma forma, contribuíram para que esta dissertação fosse escrita e mais esta etapa fosse concluída, meu muito obrigado!

“Persistence is the road to accomplishment.”

(Charles Chaplin)

RESUMO

Este trabalho apresenta o aplicativo DialogBuilder, uma ferramenta de código aberto escrita em Java que disponibiliza ao usuário uma interface para projeto de sistemas de diálogos e exportação destes para implantação no *software* Asterisk, o mais popular *framework* VoIP. O DialogBuilder disponibiliza um *wizard* para que o usuário leigo possa projetar seu sistema sem precisar aprender a programar para Asterisk. O *software* separa a fase de concepção do diálogo de sua codificação e se posiciona para tornar técnica e economicamente viável, mesmo para pequenas empresas, construir e manter sistemas de diálogo para aplicações telefônicas.

Palavras-chave: Processamento de voz. Sistema de diálogo. *Call center*. Asterisk.

ABSTRACT

This work presents DialogBuilder, an open-source Java tool that provides to its user an interface for designing dialog systems and exporting them to code that can be deployed in Asterisk, the most popular VoIP framework. DialogBuilder offers to the novice users a wizard so that they can design their own dialog system without learning the intricacies of programming for Asterisk. The software separates the stages of dialog conception and its programming and is positioned to make technically and economically viable, even for small businesses, to construct and maintain dialog systems for telephony applications.

Keywords: Speech processing. Dialog system. Call center. Asterisk.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

AGI *Asterisk Gateway Interface*
Tradução: Interface de Interligação do Asterisk

AIML *Artificial Intelligence Markup Language*
Tradução: Linguagem de Marcação para Inteligência Artificial

API *Application Programming Interface*
Tradução: Interface para Programação de Aplicativos

ASR *Automatic Speech Recognition*
Tradução: Reconhecimento Automático de Voz

CETUC *Centro de Estudos em Telecomunicações*

CPF *Cadastro de Pessoa Física*

DM *Dialog Manager*
Tradução: Gerenciador de Diálogo

EAGI *Extended Asterisk Gateway Interface*
Tradução: Interface Estendida de Interligação do Asterisk

EPBX *Electronic Private Branch Exchange*
Tradução: Comutação Eletrônica de Ramais Privados

G2P *Grapheme to Phoneme*
Tradução: Grafema-Fone

GSM *Global System for Mobile Communications*
Tradução: Sistema Global para Comunicações Móveis

HMM *Hidden Markov Model*
Tradução: Modelos Ocultos de Markov

HTML *Hypertext Markup Language*
Tradução: Linguagem de Marcação para Hipertextos

IAX *Inter-Asterisk Exchange*
Tradução: Comutação Interna do Asterisk

IDE *Integrated Development Environment*
Tradução: Ambiente de Desenvolvimento Integrado

IP *Internet Protocol*
Tradução: Protocolo de Internet

IVR *Interactive Voice Response*
Tradução: Unidade de Resposta Audível

JDK *Java Development Kit*
Tradução: Kit de Desenvolvimento Java

JSAPI *Java Speech Application Programming Interface*
Tradução: Interface Java para Programação de Aplicativos de Voz

MAP *Maximum a Posteriori*
Tradução: Máximo Posteriormente

MP3 *MPEG-1 Layer-3*
Tradução: Camada Três do Padrão de Compressão de Áudio MPEG-1

MFCC *Mel-Frequency Cepstral Coefficients*
Tradução: Coeficientes Cepstrais de Frequência da Escala Mel

MLLR *Maximum Likelihood Linear Regression*
Tradução: Regressão Linear de Maior Probabilidade

MS *Microsoft* ®

OGI *Oregon Graduate Institute*
Tradução: Instituto de Graduação de Oregon

PABX *Private Automatic Branch Exchange*
Tradução: Comutação Automática de Ramais Privados

PB *Português Brasileiro*

PBX *Private Branch Exchange*
Tradução: Comutação de Ramais Privados

POMDP *Partially Observable Markov Decision Process*
Tradução: Processo de Decisão de Markov Parcialmente Observável

PSTN *Public Switched Telephone Network*
Tradução: Rede Pública de Telefonia Comutada

SAPI *Speech Application Programming Interface*
Tradução: Interface para Programação de Aplicativos de Voz

SIP *Session Initiation Protocol*
Tradução: Protocolo de Iniciação de Sessão

TDM *Time Division Multiplexing*
Tradução: Multiplexação por Divisão do Tempo

TTS *Text to Speech*
Tradução: Síntese de Voz

UFPA *Universidade Federal do Pará*

VOIP *Voice over Internet Protocol*
Tradução: Voz sobre IP

W3C *World Wide Web Consortium*
Tradução: Consórcio da Rede Mundial de Computadores

WAV *Waveform Audio File Format*
Tradução: Formato de Arquivo de Áudio em Formas de Onda

WER *Word Error Rate*
Tradução: Taxa de Erro por Palavra

XML Extensible Markup Language
Tradução: Linguagem de Marcação Extensível

XRT *Real-time Factor*
Tradução: Fator de Tempo Real

LISTA DE SÍMBOLOS

$\boldsymbol{w}$	*Palavra*
$\boldsymbol{\tau}$	*Sentença de Palavras*
$\boldsymbol{T}$	*Conjunto de Sentenças*
$\boldsymbol{WER}$	*Taxa de Erro por Palavra*
$\boldsymbol{W}$	*Número de Palavras*
$\boldsymbol{D}$	*Número de Erros de Deleção*
$\boldsymbol{R}$	*Número de Erros de Substituição*
$\boldsymbol{xRT}$	*Fator de Tempo Real*
$\boldsymbol{PP}$	*Perplexidade*
$\boldsymbol{H_p}$	*Entropia Cruzada*

SUMÁRIO

1 INTRODUÇÃO

1.1 CONSIDERAÇÕES INICIAIS

O mundo, hoje, depende de computadores. Seja na área da saúde, educação, negócios, finanças, entretenimento, comunicação, entre outras, os mais diversos dispositivos computacionais são indispensáveis: *desktops*, *laptops*, *tablets*, *smart tvs, smartphones*, entre outros. E essa dependência fica mais acentuada quando se considera o efeito da Internet, que diminui virtualmente a distância entre as pessoas e leva informação rapidamente a qualquer um que a procure.

Dentro desse contexto, a interface entre o homem e a máquina sempre foi de suma importância. O que antes já foi feito através de cartões perfurados passou a ser feito, com o tempo, através de teclados, *mouses* e tecnologia *touch screen*, em que a tela dos dispositivos é sensível ao toque. Essa evolução busca tornar essa interação a mais simples e natural possível. E nada seria mais natural que o cenário no qual o homem pudesse conversar com a máquina em sua língua nativa, como se estivesse conversando com outro ser humano.

Sonha-se atingir esse cenário através de uma área da inteligência artificial denominada Processamento de Linguagem Natural (KAPETANIOS et al., 2013), na qual se busca exatamente permitir que o computador possa compreender corretamente a linguagem naturalmente falada por um usuário, processá-la adequadamente e responder também de forma natural, como se fosse um ser humano.

O processamento de linguagem natural se apoia fortemente na tecnologia de reconhecimento automático de voz (HUANG et al., 2001), que visa captar a fala do usuário e transformá-la em texto, sendo tema recorrente de diversas pesquisas atuais que buscam melhorá-la (HERMANSKY et al., 2013) (LEE; SINISCALCHI, 2013) (DENG; LI, 2013) (GOMEZ et al., 2013) (FOSLER-LUSSIER et al., 2013); e na tecnologia de síntese de voz (TAYLOR, 2009), que visa converter texto em fala sintetizada e que também é constante objeto de pesquisas (TOKUDA et al. 2013)

(KARHILA et al., 2013) (HUANG et al., 2013) (CHEN et al., 2013) (KASTNER; STANGL, 2013).

Sistemas de computador que se utilizam do processamento de linguagem natural para manter uma conversação com o usuário são denominados sistemas de diálogos ou agentes conversacionais (JURAFSKY; MARTIN, 2008) (ZUE; GLASS, 2000) e também são constante alvo de pesquisas recentes (LEE et al., 2013) (YOSHINO et al., 2013) (YOUNG et al., 2013) (LOPES et al., 2013) (JABAIAN et al., 2013). Existem, hoje, inúmeras aplicações para esse tipo de sistema, dentre as quais se destaca o seu uso nos *call centers* de empresas.

Call center é o nome inglês largamente utilizado para as centrais de atendimento existentes em empresas, cujo objetivo é recepcionar de maneira centralizada as chamadas de clientes para os mais diversos fins. Tais chamadas são atendidas, analisadas e distribuídas de acordo com a necessidade do atendimento em questão. Esses centros podem variar desde um pequeno grupo de funcionários que se responsabiliza por atender às ligações, entender a necessidade dos clientes e redirecionar as ligações para o destinatário correto, até grandes infraestruturas de *hardware* e *software* que objetivam automatizar o serviço além de oferecer uma gama de serviços adicionais como monitoração e acompanhamento de chamadas (SUBRAMANIAM, 2008).

Prover um primeiro diálogo com um cliente através de um *software*, obter as informações básicas para o atendimento, adiantar a solução ou, se possível, resolver o problema do cliente e, por último, quando necessário, redirecioná-lo para o destino correto é um dos serviços de um *call center* que apresenta grande retorno de investimento, sendo de interesse para qualquer empresa, seja de pequeno ou grande porte. E justamente aqui está a importância dos sistemas de diálogos, pois a eficiência do atendimento nos *call centers* está intimamente ligada à capacidade de processamento de linguagem natural desse tipo de sistema (BENNINGTON et al., 2000).

Infelizmente, a implantação de um *call center* envolve custos elevados para as empresas (BOLTE; FLEISCHMAN, 2007) e não existem ferramentas simples para que elas próprias possam projetar e implantar seus sistemas de diálogos, reduzindo assim custos, mesmo quando já existem ferramentas como o Asterisk, que é uma

ferramenta que implementa em *software* os recursos de um PABX convencional, sendo uma poderosa e eficiente arquitetura de voz sobre IP (QADEER; IMRAN, 2008) (ISEKI et al., 2011) (QIN, 2011), e que fornece, inclusive, meios gratuitos para implementação e hospedagem de tais tipos de aplicação para uso em *call centers*.

1.2 REVISÃO BIBLIOGRÁFICA

Frente ao cenário discutido, existem diversos trabalhos acadêmicos que buscam estudar a melhor forma de projetar sistemas de diálogos e também a melhor forma de avaliar sua eficácia, buscando contribuir para que as aplicações necessárias ao mundo real, com as de um *call center*, sejam cada vez mais úteis e satisfatórias aos usuários.

Kamm e Walker (1997) estudaram o projeto deste tipo de sistema e explicaram como alguns dos princípios bem entendidos e utilizados em projetos de interfaces gráficas de usuário poderiam também ser aplicados em sistemas de diálogos. Além disso, eles propuseram uma forma de avaliar tais sistemas buscando um meio de poder levar as lições aprendidas em um projeto para os próximos.

Pouco depois, Sadek (1999) analisou diversas formas de projetar um sistema de diálogos, tecendo considerações pertinentes, e propôs uma abordagem no intuito de trazer mais inteligência para tais sistemas. Scheffler (2002) também estudou os processos de projeto, avaliação e otimização de sistemas de diálogos e investigou formas de automatizar o processo de projetar tais tipos de sistemas, no intuito de acelerar seu desenvolvimento, mas mantendo a qualidade.

Sonntag et al. (2010) discutiram o projeto e implementação de um sistema de diálogo para auxiliar radiologistas a obterem anotações, imagens e outras informações relativas a seus pacientes. Forbes-Riley e Litman (2011) projetaram e avaliaram um sistema de diálogo para tutoria de alunos, em forma de *wizard*, onde as perguntas aos alunos não eram sempre as mesmas. E Benyon et al. (2013) discutiram métodos de avaliação de um agente conversacional que pode interagir

com os seres humanos em diálogos livres sobre assuntos relacionados a um dia típico de trabalho, abordando aspectos funcionais, semânticos e emocionais.

Como resultado de tais esforços acadêmicos, algumas iniciativas tecnológicas foram empreendidas no intuito de facilitar o projeto de sistemas de diálogos para aplicações no estilo das utilizadas em *call centers*.

A empresa Nuance desenvolveu os chamados Nuance Dialog Modules[1], que são blocos de código reutilizável para aplicações usando VoiceXML, que é a linguagem padronizada pelo *World Wide Web Consortium* (W3C) para aplicações de voz. Os blocos são comercializados pela empresa e possuem funcionalidades como identificação de número de cartão de crédito, de datas, de códigos postais, de e-mails, de endereços, entre outros.

A empresa Apstel desenvolveu o Visual Dialplan[2], uma ferramenta comercial gráfica para criação de sistemas de diálogos para Asterisk, disponibilizando para venda, inclusive, um servidor integrado a ser implantado junto com o Asterisk para que se possam utilizar módulos com funcionalidades especiais como acesso a outras bases de dados ou a servidores de e-mails.

No mundo dos *softwares* de código aberto, a comunidade Eclipse criou o Voice Tools Project[3], uma iniciativa para pesquisa e desenvolvimento de um *plugin* para seu ambiente de desenvolvimento integrado que propiciasse ao usuário uma ferramenta gráfica de projeto de sistemas de diálogos utilizando VoiceXML. O *software* gera, a partir do fluxo projetado, uma aplicação a ser implantada em um servidor que suporte aplicações neste padrão da W3C.

Aplicações usando VoiceXML são propícias para serem acessadas em um *Voice Browser*, em um fluxo semelhante à navegação na Web. É possível seu uso em um ambiente de *call center*, mas é preciso um esforço adicional para manter, além da estrutura da central de telefonia, um servidor de voz para hospedar a aplicação VoiceXML e uma espécie de *proxy* entre a estrutura telefônica e tal servidor (o que funcionaria como um *Voice Browser*). Já o *software* Asterisk, além de

---

[1] http://www.nuance.com/for-business/by-solution/customer-service-solutions/solutions-services/inbound-solutions/self-service-automation/dialog-modules/index.htm. Acesso em: 03 dez. 2013.
[2] http://www.apstel.com. Acesso em: 03 dez. 2013.
[3] http://www.eclipse.org/vtp. Acesso em: 03 dez. 2013.

disponibilizar toda a infraestrutura necessária para um *call center*, ainda disponibiliza uma arquitetura própria para sistemas de diálogos integrado às chamadas. É, portanto, mais natural se utilizar tal arquitetura em sistemas de diálogos para *call center*.

Como se identificou na revisão bibliográfica, muitas ferramentas, tanto comerciais quanto de código aberto, que buscam auxiliar o projeto de sistemas de diálogo, focam na linguagem padronizada VoiceXML, que é mais difícil de ser usada em um *call center* baseado em Asterisk. Já especificamente para o Asterisk, somente se identificou o Visual Dialplan, que é uma solução comercial, implicando custos imediatos para seu uso.

Neste cenário, e visando contribuir para que as próprias empresas possam projetar e implantar seus sistemas de diálogos a baixo custo, este trabalho apresenta um aplicativo de código aberto, denominado DialogBuilder, que busca disponibilizar, tal qual o Visual Dialplan, uma ferramenta gráfica para projeto de sistemas de diálogos a serem implantados no Asterisk. Além disso, o DialogBuilder possui também, como diferencial, um *wizard* em forma de diálogo que busca conduzir o usuário no processo de criação de seu projeto de sistema de diálogo, orientando-o nesta tarefa e possibilitando que mesmo um usuário leigo possa projetar seu sistema de acordo com o que precisa, sem necessidade de prévio conhecimento técnico.

1.3 DIRETRIZES DE PESQUISA

Considerando o exposto, bem como a importância de se trabalhar continuamente para melhorar a qualidade dos *call centers* existentes nas empresas, este trabalho desenvolveu pesquisa acerca dos sistemas de diálogos tipicamente utilizados em tais centrais e, aliada a pesquisas desenvolvidas sobre uso de reconhecimento de fala em aplicações de *software*, seus resultados foram utilizados no desenvolvimento de um aplicativo que torna possível que o próprio usuário possa projetar e construir um sistema de diálogo para *call center* baseado na arquitetura do

software Asterisk. O intuito é que esse aplicativo evolua e seja capaz de prover suporte a sofisticadas estratégias de processamento de linguagem natural em ambientes de *call centers*.

As diretrizes usadas no desenvolvimento da pesquisa realizada foram alinhadas à busca de um aplicativo com as seguintes características, devidamente justificadas:

- **Apresentar uma interface *desktop* em Java:** como o projeto e construção do sistema de diálogo pode ser feito de maneira local, sem necessidade de conexão com a Internet, um aplicativo *desktop* é o mais indicado e a linguagem Java foi escolhida por ser de código aberto, flexível e com recursos adequados para o desenvolvimento;

- **Possuir funcionalidade para criar, editar, excluir e consultar projetos de sistema de diálogo:** para que os diversos projetos possam ser mantidos na aplicação, são necessárias as funcionalidades de criação, edição, exclusão e consulta de tais projetos;

- **Possuir uma ferramenta de visualização gráfica do fluxo de diálogo de um projeto, permitindo sua edição de maneira gráfica:** para que o próprio usuário possa projetar seu sistema de diálogo, é necessário prover meios para que isso seja facilitado e a melhor opção é permitir que o usuário seja capaz de realizar esta tarefa de forma gráfica, que é mais intuitiva para os seres humanos;

- **Ter os seguintes módulos disponíveis para uso no fluxo de diálogo de um projeto: “Play Prompt” (execução de uma mídia), “Simple Question” (pergunta simples), “Log” (registro de resposta do cliente), “Multiple Question” (pergunta com múltiplas respostas), “Yes/No Question” (pergunta do tipo “sim ou não”) e “Flow End” (encerramento da chamada):** as ações de fluxo acima são muito comuns e elementares em sistemas de diálogos e, portanto, foram selecionadas para serem implementadas em módulos que ficarão disponíveis para uso

no projeto dos usuários; além disso, tais ações são suficientes para projetar diversos sistemas de diálogos simples, porém funcionais;

- **Possuir funcionalidade para exportar o projeto de sistema de diálogo, gerando todos os insumos necessários para implantar tal projeto no *software* Asterisk:** quando o projeto do sistema de diálogo estiver concluído, é necessário construir o sistema para uso no Asterisk e, portanto, tal funcionalidade se faz necessária;

- **Possuir um tratamento mínimo de internacionalização:** a ideia é que o aplicativo possa ser utilizado por usuários que falem quaisquer línguas e, para isso, é necessário tratar da internacionalização do sistema; como o tratamento completo envolve muitas etapas, optou-se por apenas iniciar esta internacionalização por ora;

- **Possuir um *wizard* em forma de diálogo falado, possibilitando que o usuário possa projetar um sistema de diálogo através de uma conversa com o aplicativo, utilizando todos os módulos disponibilizados neste:** a ideia é buscar uma forma de guiar o usuário no projeto do sistema de diálogo, facilitando ainda mais esta tarefa para ele; optou-se então por um assistente *wizard* em forma de diálogo, pois se aposta que esta é a forma mais natural e eficiente para este guia;

- **Estar apto para que novos módulos possam ser criados e incluídos, inclusive no *wizard*:** como se pretende evoluir constantemente o aplicativo, é muito importante que ele seja desenvolvido de forma a poder facilmente incorporar novos módulos, seja no método gráfico de projeto ou no *wizard*;

- **Apresentar instruções para que o sistema de diálogo projetado possa ser implantado no *software* Asterisk:** é importante que o usuário saiba exatamente como proceder para implantar, no Asterisk, o sistema de diálogo gerado pelo aplicativo; portanto, esta funcionalidade se faz necessária.

1.4 PUBLICAÇÃO GERADA PELA PESQUISA

Os resultados obtidos com a pesquisa realizada foram publicados no seguinte trabalho científico:

BORGES FILHO, E. L. M; BATISTA, P; KLAUTAU, A. Uma Ferramenta para Projeto de Sistemas de Diálogos para Call Center Baseados em Asterisk. In: BRAZILIAN SYMPOSIUM IN INFORMATION AND HUMAN LANGUAGE TECHNOLOGY, 9th, 2013, Fortaleza. **Proceedings…** Fortaleza: [s.n.], 2013. p.225-229.

2 CONCEITOS BÁSICOS

2.1 PROCESSAMENTO DE VOZ

Processamento de voz (ou processamento de fala) se refere às tecnologias relacionadas ao reconhecimento automático de voz (ASR), à síntese de voz (TTS) e ao uso das informações de voz (HUANG et al., 2001). ASR é o processo que visa converter um sinal de fala digitalizado em texto; TTS é o processo que busca converter texto na forma de linguagem natural em fala sintetizada. O uso das informações de voz se refere às diversas aplicações que podem ser criadas utilizando esta tecnologia e, basicamente, busca mapear o texto reconhecido em ações do sistema. Um exemplo é utilizar voz para comandar um navegador de Internet para deficientes visuais (BATISTA et al., 2010).

2.1.1 Síntese de Voz

A síntese de voz é o processo de produção artificial da voz humana, tipicamente convertendo uma representação textual para fala num formato de áudio. O funcionamento básico de um sistema de síntese de voz consiste de duas partes: um *front-end* e um *back-end*.

O *front-end* possui duas funções principais: primeiro, ele converte símbolos textuais, como números ou abreviações, em palavras por extenso. Este processo é conhecido como normalização textual ou pré-processamento. Em seguida, cada palavra é transcrita foneticamente e o texto é dividido em unidades prosódicas como frases, cláusulas ou sentenças. Este processo é conhecido como *text-to-phoneme*. O conjunto das transcrições fonéticas junto com as informações prosódicas forma a

representação linguística simbólica do texto, que é a saída do *front-end* do sintetizador de voz.

O *back-end* é responsável por converter a representação linguística simbólica em som. Este processo pode ser realizado de diferentes formas. Se o domínio do problema é restrito, isto é, se há um prévio conhecimento do conjunto de possíveis textos a serem sintetizados, uma técnica que possível é a gravação, onde o texto a ser sintetizado é previamente gravado em um arquivo de áudio. Esta técnica pode ser melhorada através da concatenação, onde se gravam previamente unidades textuais (palavras, fonemas, difonemas, entre outros) e o sintetizador as concatena de acordo com a necessidade. A aplicação de tais técnicas fica, claramente, mais difícil quanto maior e mais livre é o conjunto de textos passíveis de sintetização. Além disso, apesar deste tipo de técnica utilizar sons muito naturais (gravados por serem humanos), as diferenças de entonação nas gravações e as técnicas automatizadas para concatená-los podem trazer falhas no resultado final.

Outra técnica existente é através de formantes. Esta técnica utiliza um modelo acústico e, de fato, cria voz sintética, através da variação de parâmetros como frequência fundamental, afinação e nível de ruído das formas de onda. O resultado desta técnica é muitas vezes uma voz pouco natural, claramente diferente de uma voz humana, mas nem sempre isso é um problema. Além disso, sintetizadores que utilizam formantes são bem menores e geram discursos sem os *gaps* existentes em modelos de concatenação. Também é possível combinar as técnicas para melhores resultados. Outra técnica que também é possível utiliza modelos ocultos de Markov (HMM) para modelar estatisticamente o espectro de frequências, a frequência fundamental e a duração da fala, sendo a saída do modelo as formas de onda do discurso sintetizado. Além das citadas, existem ainda diversas outras abordagens para a implementação do *back-end* de um sintetizador de voz.

Cada uma das etapas acima possui diversos desafios que devem ser levados em consideração, como a inclusão de emoção no discurso gerado (uma notícia triste não pode ser falada de forma animada, por exemplo), a variação na forma de falar numerais (falar um número de telefone é diferente de falar um numeral por extenso, por exemplo) ou a diferença de fonemas para uma mesma letra ou sílaba, o que dificulta no trabalho do processo *text-to-phoneme*. Além disso, há ainda as

diferenças inerentes a cada idioma que também devem ser consideradas. A Figura 2.1 apresenta uma visão geral de um típico sistema de síntese de voz.

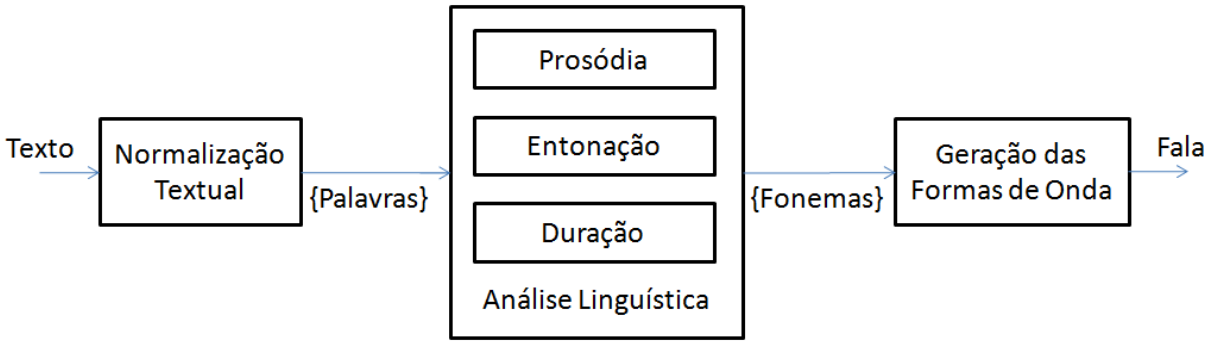


Figura 2.1 – Visão geral do processo de síntese de voz.

Taylor (2009) realiza uma análise completa do processo de sintetização de voz por computador, com explicações detalhadas de todos os aspectos envolvidos nas tecnologias de geração de fala.

Existem no mercado diversos sintetizadores de voz, tanto comerciais, como o IVONA Reader[4], quanto de código aberto, como FreeTTS[5]. Alguns, inclusive, com suporte ao Português Brasileiro, como o *software* livre eSpeak[6].

2.1.2 Reconhecimento Automático de Voz

Um sistema ASR típico adota uma estratégia estatística baseada em modelos ocultos de Markov (HMM), detalhadamente estudados por Huang et al. (1991), e é composto por cinco blocos: *front-end*, dicionário fonético, modelo acústico, modelo de linguagem e decodificador, como indicado na Figura 2.2. As duas principais aplicações de ASR são comando e controle e ditado (HUANG et al., 2001). A

[4] http://www.ivona.com/us/reader. Acesso em: 15 nov. 2013.
[5] http://freetts.sourceforge.net. Acesso em: 15 nov. 2013.
[6] http://espeak.sourceforge.net. Acesso em: 15 nov. 2013.

primeira é relativamente simples, pois o modelo de linguagem é composto por uma gramática que restringe as sequências de palavras aceitas. A última, tipicamente, suporta um vocabulário com mais de 60 mil palavras e exige mais processamento.

Segundo Huang et al. (2001), o processo de *front-end* convencional extrai segmentos curtos (janelas ou *frames*) de um sinal de voz e converte, a uma taxa de *frames* constante, cada segmento para um vetor $\boldsymbol{x}$ de dimensão L. Assume-se aqui que T *frames* são organizados em uma matriz $\boldsymbol{X}$ de dimensão $L \times T$, que representa uma sentença completa. Existem várias alternativas no que diz respeito à parametrização do sinal de voz. Apesar da análise dos coeficientes cepstrais de frequência da escala Mel (MFCC) ser relativamente antiga (DAVIS; MERMELSTEIN, 1980), essa provou ser efetiva e é geralmente usada como entrada para os blocos de *back-end* do sistema ASR (HUANG et al., 2001).

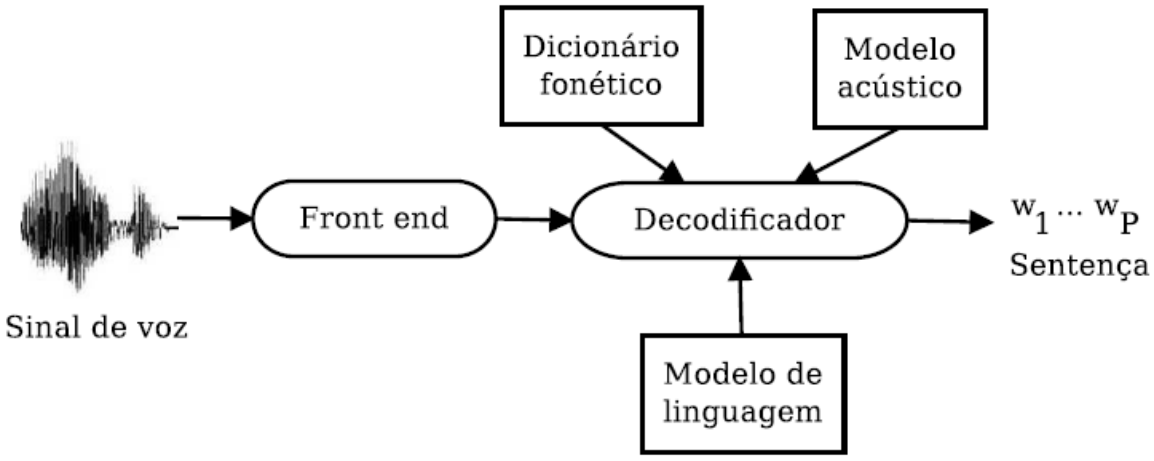


Figura 2.2 – Principais blocos de um típico sistema ASR.

O modelo de linguagem de um sistema de ditado, fornece a probabilidade $p(\tau)$ de observar a sentença $\tau = [w_1, \ldots, w_P]$ de P palavras. Segundo Rabiner e Juang (1993), conceitualmente, o decodificador tem como objetivo achar as sentenças τ^* que maximizam a probabilidade *a posterior* dada por:

$$\tau^* = arg\max_{\tau} p(\tau|\boldsymbol{X}) = arg\max_{\tau} \frac{p(\boldsymbol{X}|\tau)p(\tau)}{p(\boldsymbol{X})} \tag{2.1}$$

onde $p(\boldsymbol{X}|\tau)$ é dada pelo modelo acústico. Como $p(\boldsymbol{X})$ não depende de τ, a equação anterior é equivalente a:

$$\tau^* = arg\max_\tau p(\boldsymbol{X}|\tau)p(\tau) \quad (2.2)$$

Na prática, uma constante empírica é usada para ponderar a probabilidade do modelo de linguagem $p(\tau)$ antes de a mesma ser combinada com a probabilidade dos modelos acústicos $p(\boldsymbol{X}|\tau)$.

Dado o grande volume de sentenças concorrentes (hipóteses), a Equação 2.2 não pode ser calculada independentemente para cada hipótese. Portanto, os sistemas ASR usam estruturas de dados como árvores léxicas, usando o artifício de separar as sentenças em palavras, e as palavras em unidades básicas, aqui chamadas de fones (HUANG et al., 2001). A busca por τ^* é chamada decodificação e, na maioria dos casos, hipóteses são descartadas ou podadas (*pruning*). Em outras palavras, para tornar viável a busca pela "melhor" sentença, algumas candidatas são descartadas e a Equação 2.2 não é calculada para elas (DESHMUKH et al., 1999) (JEVTIC et al., 2001).

Um dicionário fonético (conhecido também como modelo léxico) faz o mapeamento das palavras em unidades básicas (fones) e vice-versa. Para um melhor desempenho, HMM contínuas são adotadas, onde a distribuição de saída de cada estado é modelada por uma mistura de Gaussianas, como mostrado na Figura 2.3. A topologia típica de uma HMM é a esquerda-direita, onde as únicas transições permitidas são de um estado para ele mesmo ou para o estado seguinte.

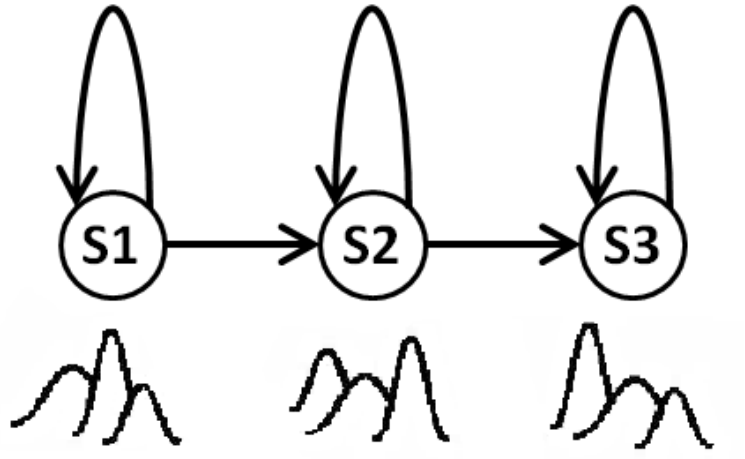


Figura 2.3 – Representação gráfica de uma HMM contínua de topologia esquerda-direita com três estados e uma mistura de três Gaussianas por estado. Adaptada de Batista (2013).

De acordo com Huang et al. (2001), dois problemas clássicos com relação à modelagem acústica são: a inconstância dos fones devido à coarticulação e a insuficiência de dados para estimar os modelos. O método de compartilhamento de parâmetros (*sharing*) visa combater esse último problema, melhorando a robustez dos modelos. Em muitos sistemas, o compartilhamento é implementado no nível de estado, ou seja, o mesmo estado pode ser compartilhado por HMM diferentes.

O modelo acústico pode conter uma HMM por fone. O que seria uma boa suposição caso um fone pudesse ser seguido por qualquer outro, o que não é verdade, já que os articuladores do trato vocal não se movem de uma posição para outra imediatamente na maioria das transições de fones. Nesse sentido, durante o processo de criação de sistemas que modelam a fala fluente, busca-se um meio de modelar os efeitos contextuais causados pelas diferentes maneiras que alguns fones podem ser pronunciados em sequência (LADEFOGED, 2001). A solução encontrada é o uso de HMM dependentes de contexto, que modelam o fato de um fone sofrer influência dos fones vizinhos. Por exemplo, supondo a notação do trifone $a - b + c$, tem-se que b representa o fone central ocorrendo após o fone a e antes do fone c.

Segundo Huang et al. (2001), existem basicamente dois tipos de modelos trifones: *internal-word* e *cross-word*. As diferenças entre os mesmos é que no caso do *internal-word* as coarticulações que extrapolam as durações das palavras não são consideradas, o que implica que menos modelos são necessários. Já no caso do *cross-word*, que considera a coarticulação entre o final de uma palavra e o início da seguinte, a modelagem é mais precisa, porém o número de modelos trifones gerados cresce muito, o que dificulta o trabalho do decodificador e gera uma necessidade de mais dados para treino. Alguns exemplos de transcrição podem ser conferidos na Tabela 2.1.

Tabela 2.1 – Exemplos de transcrições com modelos independentes e dependentes de contexto.

Sentença	arroz com bife
Monofones	sil a R o s k o~ b i f i sil
Internal-Word	sil a+R a-R+o R-o+s o-s k+o~ k-o~ b+i b-i+f i-f+i f-i sil
Cross-Word	sil sil-a+R a-R+o R-o+s o-s+k s-k+o~ k-o~+b o~-b+i b-i+f i-f+i f-i+sil sil

Segundo Rabiner e Juang (1993), a escassez de dados de treino também afeta a modelagem da língua, pois:

$$P(\tau) = P(w_1, w_2, \dots, w_P) \Rightarrow \tag{2.3}$$

$$P(\tau) = P(w_1)P(w_2|w_1) \dots P(w_P|w_1, w_2, \dots w_{P-1}) \tag{2.4}$$

É impraticável estimar a probabilidade condicional $P(w_i|w_1, w_2, \dots w_{i-1})$, mesmo para valores moderados de i. Assim, o modelo de linguagem para sistemas ASR consiste de um modelo n-gram, que assume que a probabilidade $P(w_i|w_1, w_2, \dots w_{i-1})$ depende somente das $n-1$ palavras anteriores. Por exemplo, a probabilidade $P(w_i|w_{i-2}, w_{i-1})$ expressa um modelo de linguagem trigrama.

Resumindo, após o treinamento de todos os modelos estatísticos, um sistema ASR na etapa de teste usa o *front-end* para converter o sinal de entrada em parâmetros e o decodificador para encontrar a melhor sentença τ.

Os modelos acústicos e de linguagem podem ser fixos durante a fase de teste, porém adaptá-los pode gerar um melhor desempenho. Por exemplo, o domínio de aplicação pode ser estimado e um modelo de linguagem específico usado. Isso é crucial para aplicações com vocabulário técnico, como relatórios médicos de raios X (ANTONIOL et al., 1993). A adaptação do modelo acústico possui igual importância (LEE; GAUVAIN, 1993), sendo que o modelo pode ser adaptado, por exemplo, a um locutor, ou ao sotaque de uma determinada região.

Sistemas ASR que usam modelos independentes de locutor são convenientes, porém devem ser robustos o suficiente para reconhecer, com bom desempenho, áudio de qualquer locutor. Com o custo de exigir que o usuário leia algumas sentenças, técnicas de adaptação ao locutor podem melhorar os modelos HMM para um locutor específico. Técnicas de adaptação podem também ser usadas para compensar variações no ambiente acústico, reduzindo o descasamento causado pelo canal ou efeitos de ruído aditivo.

O conjunto de um decodificador e todos os recursos necessários para sua execução (modelos de linguagem e acústico, etc.) é comumente referido como *engine* ASR. Existem diversas *engines* ASR disponibilizadas comercialmente por empresas como Microsoft®, através do *Microsoft Language Development Center*

Portugal[7], e Nuance[8]. O Grupo FalaBrasil[9] do Laboratório de Processamento de Sinais da Universidade Federal do Pará (UFPA) desenvolveu uma *engine* ASR, de código aberto, específica para o Português Brasileiro, denominada Coruja (SILVA et al., 2010).

2.1.3 Avaliação do Reconhecimento Automático de Voz

Para a maioria das aplicações que usam reconhecimento de voz, incluindo ditado, a medida de desempenho utilizada é a taxa de erro por palavra (WER). Para calculá-la, dado que geralmente as transcrições correta e reconhecida possuem um número diferente de palavras, elas são primeiramente alinhadas através de programação dinâmica (BELLMAN, 1957). Após tal passo, de acordo com Huang et al. (2001), a WER pode ser calculada através da fórmula a seguir:

$$WER = \frac{D+R+I}{W} \times 100\% \tag{2.5}$$

onde W é o número de palavras na sentença de entrada e R, D e I são, respectivamente, o número de erros de substituição, deleção e inserção de palavras na sentença reconhecida quando comparada com a transcrição correta.

Outra métrica de avaliação é o fator de tempo-real (xRT). O fator xRT é calculado dividindo o tempo que o sistema despende para reconhecer uma sentença pela sua duração. Assim, quanto menor for o fator xRT, mais rápido será o reconhecimento.

Durante o processo de construção dos modelos de linguagem, existem várias características que os diferenciam, como o número de palavras distintas e o número de sentenças usadas para estimar os modelos. No entanto, a métrica mais comum para avaliar os modelos de linguagem é a probabilidade $p(\mathbf{T})$ que o modelo atribui

[7] http://www.microsoft.com/portugal. Acesso em: 17 nov. 2013.
[8] http://www.nuance.com. Acesso em: 17 nov. 2013.
[9] http://www.laps.ufpa.br/falabrasil. Acesso em: 17 nov. 2013.

para alguns dados de teste $\mathbf{T} = \{\tau_1, \dots, \tau_B\}$ compostos de B sentenças. Independência entre as sentenças é assumida, o que leva ao produto das probabilidades $p(\mathbf{T}) = p(\tau_1) \dots p(\tau_B)$. Duas medidas são derivadas dessa probabilidade: perplexidade e entropia cruzada (*cross-entropy*) (HUANG et al., 2001). A entropia cruzada $H_p(\mathbf{T})$ é definida como:

$$H_p(\mathbf{T}) = -\frac{1}{W_\mathbf{T}} \log_2 p(\mathbf{T}) \tag{2.6}$$

onde $W_\mathbf{T}$ é o número de palavras em $\mathbf{T}$.

A perplexidade (PP) representa o número médio de palavras que podem seguir uma dada palavra e está relacionada com a *cross-entropy* $H_p(\mathbf{T})$ por:

$$PP = 2^{H_p(\mathbf{T})} \tag{2.7}$$

Para uma dada tarefa (por exemplo, com o tamanho do vocabulário fixado), baixos valores de perplexidade e *cross-entropy* indicam menor incerteza na predição da próxima palavra e, tipicamente, revelam um melhor modelo de linguagem.

2.1.4 Histórico de Atividades em Processamento de Voz na UFPA

O primeiro esforço do Grupo FalaBrasil do Laboratório de Processamento de Sinais da UFPA em termos de recursos a serem disponibilizados foi a criação do UFPAdic versão 1, um dicionário fonético para Português Brasileiro (PB) com 11.827 palavras transcritas manualmente (HOSN et al., 2006). Esse conjunto de transcrições foi então utilizado como base de treino de um módulo de conversão grafema-fone (G2P), capaz de extrair automaticamente regras de transcrição de um léxico através de técnicas de aprendizado de máquina por indução (árvore de decisão) e Naïve Bayes. O resultado foi a criação e disponibilização de um novo dicionário fonético de grande vocabulário para PB com aproximadamente 60 mil palavras, chamado UFPAdic versão 2.

Neto et al. (2008a) buscaram, então, criar um sistema de referência utilizando as seguintes bases de voz: Spoltech (SCHRAMM et al., 2006) e OGI-22 (LANDER et al., 1995). O principal objetivo foi desenvolver e disponibilizar recursos para PB, a fim de criar *baselines* que permitissem a reprodução dos resultados em diferentes locais. Para isso, foram disponibilizados, na página do Grupo FalaBrasil, modelos acústicos e de linguagem que são constantemente atualizados. O dicionário UFPAdic versão 2 foi usado para a construção dos modelos ocultos de Markov (HMM) presentes nos recursos disponibilizados, a partir dos *corpus* OGI-22 e Spoltech, totalizando 32 fones e um modelo de silêncio. Em seguida, modelos trifone foram construídos a partir dos modelos monofone e uma árvore binária de decisão fonética foi elaborada para vincular os modelos trifone com as mesmas características. Nos experimentos realizados, após um trabalho de correção dos *corpora*, a base de treino do Spoltech foi composta por 5.246 arquivos de áudio, que correspondem a 180 minutos, e a base de teste com os restantes 2.000 arquivos, que totalizaram 40 minutos de gravação. Já a base de treino do OGI-22 consistiu de 2.017 arquivos de áudio, correspondendo a 184,5 minutos de gravação, e o conjunto de teste com 209 arquivos, totalizando 14 minutos.

Com a finalidade de avaliar o modelo acústico do OGI-22, o primeiro experimento consistiu na construção de modelos de linguagem bigrama utilizando as próprias frases do *corpus* OGI-22 (ou seja, uma gramática viciada). O número de Gaussianas por mistura foi gradualmente incrementado até o total de 10 componentes (limite de decréscimo da taxa de erro). A taxa de erro por palavras (WER) foi de 21% para 3.285 palavras. Da mesma forma, para avaliar o modelo acústico do Spoltech, modelos de linguagem bigrama foram construídos apenas com as frases dessa base. A WER com 10 componentes de Gaussianas por mistura foi 18,6% para 793 palavras. Em um segundo momento, frases da base de texto CETENFolha[10] foram formatadas e incluídas na construção dos modelos de linguagem. Primeiramente, para o treino dos modelos de linguagem, foram selecionadas e fixadas 32.100 frases das bases OGI-22 e CETENFolha. O vocabulário foi aos poucos incrementado com as palavras mais frequentes no conjunto de treino. Assim, vários modelos bigrama foram construídos. Usando o

[10] http://www.linguateca.pt/CETENFolha. Acesso em: 07 dez. 2013.

modelo acústico do *corpus* OGI-22 com 10 componentes de Gaussianas por mistura, a melhor WER encontrada para 30.000 palavras foi 35,87%.

Modelos trigrama e técnicas de suavização foram usadas por Silva et al. (2008) para construção de modelos de linguagem mais elaborados. O modelo acústico do OGI-22 apresentou uma WER de 23,22% e 20,40% para bigramas e trigramas viciados e suavizados via técnica Witten-Bell, respectivamente. Da mesma forma, modelos de linguagem n-grama foram construídos apenas com as frases da base Spoltech, com vocabulário de 1.104 palavras. A WER com 14 componentes de Gaussianas por mistura foi 6,13% e 5,27% para bigramas e trigramas, respectivamente, apresentando consideráveis melhoras em relação os resultados obtidos por Neto et al. (2008a).

Outro experimento, realizado por Silva et al. (2008), consistiu em avaliar o desempenho do sistema variando o número de sentenças usadas para o treino dos modelos de linguagem e, dessa vez, completamente livres de contexto, ou seja, usando apenas frases do *corpus* CETENFolha. Assim, bigramas e trigramas, com suavização Kneser-Ney, foram elaboradas variando o número de frases para treino entre 10.000 e 180.000 e o vocabulário mantido constante, contendo apenas as palavras presentes no OGI-22. A melhor taxa de erro por palavra usando o modelo acústico do *corpus* OGI-22 com 10 componentes de Gaussianas por mistura foi 47,39% para o sistema com trigrama, treinado com 180.000 sentenças.

Siravenha et al. (2008) contribuíram com um algoritmo baseado em regras para conversão G2P com determinação de vogal tônica para PB. A estrutura de regras utilizada baseou-se nas regras G2P descritas por Silva et al. (2006), sendo que algumas correções e adições foram propostas. Para avaliar a eficácia das regras G2P com determinação de vogal tônica, foi construído um dicionário fonético composto pelas 679 palavras presentes no *corpus* West Point (MORGAN et al., 2008). Em seguida, esse dicionário foi usado para treinar um modelo acústico com 38 fones. Para efeito de comparação, o mesmo procedimento foi usado para elaborar outros dois modelos acústicos. O segundo modelo desenvolvido usou um dicionário fonético fiel às regras fonológicas descritas por Silva et al. (2006), ou seja, sem considerar as alterações propostas por Siravenha et al. (2008). Já o último modelo acústico, composto por 34 HMMs, empregou o dicionário UFPAdic 2. Os melhores resultados foram obtidos com os sistemas que utilizaram um dicionário

baseado em regras. Contudo, diante de um modelo de linguagem independente do contexto, ficou mais nítido que as mudanças sugeridas às regras de Silva et al. (2006) melhoraram o desempenho do reconhecedor.

Até então, o *corpus* de texto CETENFolha e as transcrições ortográficas das bases de voz disponíveis eram utilizados para construir os modelos de linguagem usados nos experimentos. Visto que o CETENFolha não é um *corpus* tão recente, sua utilização para a tarefa de reconhecimento de voz em tempo real fica prejudicada. Assim, textos recentes de outros jornais foram adicionados ao *corpus*. Isso foi feito através de um processo totalmente automatizado de coleta e formatação diária de três jornais, disponíveis na Internet. Aproximadamente dois meses de textos coletados e processados encontram-se disponíveis na página do Grupo FalaBrasil, que correspondem a aproximadamente 120 mil frases.

Dentre as dificuldades encontradas em se produzir grandes *corpora*, tem-se a coleta de dados (voz) e transcrição ortográfica. Visando contornar tal problema, Silva et al. (2009) construíram um novo *corpus* de voz baseado em *audiobooks*. Os *audiobooks* são livros falados disponíveis na Internet. Com os arquivos de áudio e suas respectivas transcrições (livros) tem-se uma redução considerável na tarefa de produção de um *corpus*. Inicialmente, foram obtidos 5 livros com aproximadamente 1 hora de duração cada. Os arquivos de áudio foram reamostrados para 22050 Hz com 16 bits por amostra, em seguida foram divididos em arquivos menores, com 30 segundos de duração cada, e por fim transcritos. O *corpus* foi composto por 2 locutores do sexo masculino, e 3 do sexo feminino, totalizando 5 horas e 19 minutos de áudio. Um problema encontrado na utilização de *audiobooks* foi o ambiente de gravação, que é bastante controlado, fazendo com que os arquivos não possuam qualquer tipo de ruído, diferente dos arquivos de teste real, onde o ambiente acústico não é controlado. Contudo tal problema pode ser contornado com técnicas de adaptação ao ambiente (SILVA et al., 2009).

Silva et al. (2009), com o intuito de obter uma boa avaliação de desempenho e possibilitar a comparação de resultados com outros grupos de pesquisas, ainda construíram um *corpus* composto por 500 frases exclusivamente para testes. Para construção do *corpus* foram utilizadas frases retiradas do trabalho de Cirigliano et al. (2005). O *corpus* construído por Silva et al. (2009) possuía 25 locutores com 20 frases cada, sendo 17 homens e 8 mulheres, que correspondem a aproximadamente

42 minutos de áudio. Todas as gravações foram realizadas em computadores pessoais utilizando microfones comuns, a taxa de amostragem utilizada foi de 22050 Hz com 16 bits por amostra. O ambiente não foi controlado, existindo assim a presença de ruído nas gravações, caracterizando ambientes onde *softwares* de reconhecimento de voz são utilizados.

Nos primeiros trabalhos, buscou-se criar um sistema de referência utilizando as bases de voz: Spoltech, OGI-22 e West Point. Porém, o sistema se limitava a um reconhecimento “controlado”, ou seja, treino e teste com características semelhantes (locutores, ambiente de gravação, etc.). Já no trabalho de Silva et al. (2009), o objetivo foi criar um sistema apto a trabalhar em tempo real utilizando o decodificador HDecode do pacote HTK (YOUNG et al., 2006). Foram construídos modelos acústicos e de linguagem voltados para tal tarefa. Além disso, após as etapas de construção dos modelos, técnicas de adaptação ao ambiente foram utilizadas visando melhorar o desempenho do sistema. Assim, modelos acústicos utilizando 14 Gaussianas por mistura e trifones dependentes de contexto foram criados utilizando a base de voz Spoltech e os *audiobooks*. Um modelo de linguagem trigrama com perplexidade igual a 169 foi elaborado utilizando 1,6 milhões de frases retiradas das bases: CETENFolha, Spoltech, OGI-22, West Point e *audiobooks*; além dos textos recentemente extraídos da Internet. Todas as simulações foram realizadas com o novo *corpus* de teste e com peso do modelo acústico igual a 2 no processo de decodificação.

O melhor resultado foi obtido com o modelo acústico treinado somente com o Spoltech, onde a WER foi igual a 44,3% e fator de tempo-real (xRT) igual a 1,5. O pior resultado foi obtido com o modelo acústico treinado apenas com os *audiobooks*, com 55,9% de WER e 3 em xRT. Resultados esperados, já que os *audiobooks* foram produzidos em um ambiente acústico totalmente diferente do *corpus* de teste (ausência de ruído). Diante disso, buscou-se através de técnicas de adaptação ao ambiente, aumentar a precisão do sistema. Com o auxílio da ferramenta HTK, foram utilizadas duas técnicas de adaptação: *maximum a posteriori* (MAP) e *maximum likelihood linear regression* (MLLR). Partindo do modelo acústico produzido a partir dos *audiobooks*, fez-se uma adaptação (introdução de ruído) utilizando o *corpus* Spoltech. Em geral, a combinação MLLR e MAP tende a apresentar melhores

resultados, fato que se confirmou, já que o melhor resultado foi obtido combinando as duas técnicas: 29,6% de WER e 2,4 em xRT.

Outro campo de pesquisa concentra-se no estudo e desenvolvimento de aplicativos com interface aural a partir da Microsoft Speech Application Programming Interface (SAPI). Por exemplo, Neto et al. (2008b) apresentaram um *software* para aprendizado de língua inglesa com o auxílio do computador (CALL). Resumidamente, o aluno é induzido a responder oralmente, ou mesmo manualmente, os questionamentos através de estímulos escritos, falados e visuais. Além dos exercícios objetivos e subjetivos propostos, existe a possibilidade de um treinamento prévio, onde o aluno pode enriquecer seu vocabulário escutando, via síntese de voz, palavras individualizadas, ou frases, sempre com o respectivo retorno visual, ilustrando o significado da palavra, ou a ação associada à frase.

O aplicativo CALL é capaz de interagir com o usuário através das línguas portuguesa (síntese de voz) e inglesa (síntese e reconhecimento de voz) utilizando os seguintes *engines* providos pela Microsoft®: Microsoft English Recognizer e Lernout & Hauspie TTS. Outra funcionalidade é o agente personalizado, Merlin, criado a partir do componente Microsoft Speech Agent para prover *feedback* e assistência quando necessário. Posteriormente, no trabalho de Siravenha et al. (2009), o reconhecedor de voz Microsoft Speech Recognition Sample Engine for Portuguese (Brazil), em sua versão beta, foi incorporado ao aplicativo CALL, acrescentando interface de reconhecimento de voz em PB.

Em outro trabalho, Batista et al. (2010) discutiram a integração de diferentes tecnologias disponíveis para o desenvolvimento de sistemas de diálogo falado em PB. Como exemplo, o trabalho apresenta o protótipo de um sistema para a navegação não visual na Web, em ambiente Windows. Com base na interface SAPI, o sistema estabelece um diálogo falado com o usuário, questionando-o sobre o *site* e a palavra que deseja consultar. Como resposta, o conteúdo principal da página resultada da busca pelo usuário é lido. O próprio sistema coordena as interações com o usuário e, atualmente, é limitado à busca pelo nome de países.

Em 2009, com o intuito de promover o amplo desenvolvimento de aplicações com suporte a reconhecimento de voz em PB, o Grupo FalaBrasil disponibilizou à comunidade de desenvolvedores seu sistema para reconhecimento automático de

voz, o Coruja (SILVA et al., 2010), um *engine* de voz específico para o PB. Esse *software* é composto pelo decodificador livre Julius; modelo acústico e de linguagem para o reconhecimento de voz em PB; e uma API própria, implementada na linguagem de programação C/C++ chamada LaPSAPI. Essa API pode ser utilizada tanto na plataforma Linux, a partir de implementações em C++, quanto na plataforma Windows, através de qualquer linguagem da plataforma Microsoft .NET (C#, Visual Basic, entre outras), garantindo certa flexibilidade tanto quanto à plataforma, quanto à linguagem de programação.

Posteriormente, com o objetivo de tornar o Coruja um recurso mais abrangente no que diz respeito às plataformas de desenvolvimento suportadas, no trabalho de Oliveira et al. (2011), uma API compatível com a especificação JSAPI da Sun Microsystems, chamada JLaPSAPI, foi desenvolvida e disponibilizada à comunidade como parte do Coruja. Com a adição desta API ao Coruja, tornou-se possível o desenvolvimento de aplicativos com suporte a reconhecimento de voz através da linguagem de programação Java a qual até então, o Coruja não oferecia suporte.

Em trabalho recente, Oliveira et al. (2012) construíram modelos acústicos para o PB utilizando o pacote de ferramentas CMU Sphinx. Esse trabalho foi desenvolvido tendo como objetivo principal gerar recursos que permitissem o desenvolvimento de aplicativos com suporte a reconhecimento de voz em plataformas móveis, como Android e IOs. Os modelos acústicos foram treinados utilizando parte da base West Point, somada a LaPSStory e a base de áudio do Centro de Estudos em Telecomunicações (CETUC), totalizando aproximadamente 143 horas de áudio. Para os testes referentes à tarefa de ditado, a base LaPSBenchmark foi utilizada. O melhor resultado obtido para ditado, com WER de 16,41% e xRT em torno de 1, foi obtido pelo modelo acústico com 2.000 estados vinculados e 8 Gaussianas por mistura. Já para a tarefa de comando e controle, aferida no contexto de uma aplicação de correio eletrônico utilizando a base LaPSMail, o melhor resultado, com WER de 0,67%, foi obtido pelo modelo acústico com 500 estados vinculados e 64 Gaussianas por mistura.

Além disso, Batista et al. (2012) desenvolveram uma extensão da ferramenta Writer do pacote LibreOffice, denominada SpeechOO. O intuito da ferramenta é

aumentar o nível de acessibilidade do editor de textos, tornando possível que pessoas com algum tipo de dificuldade motora possam elaborar textos usando ditado e formatar o texto ou controlar o Writer através de comandos de voz.

Recentemente, Batista et al. (2013) compararam três sistemas de reconhecimento de fala que podem ser usados no desenvolvimento de aplicativos para o sistema operacional Android: PocketSphinx, utilizado localmente; Julius em modo servidor, com reconhecimento distribuído via Internet através da *cloud* do Grupo FalaBrasil; e Google, também distribuído. Eles constataram que o uso de memória e CPU do PocketSphinx é bem maior que o do Julius e do Google, que ficaram próximos em ambos os quesitos. Quanto ao WER e xRT, o PocketSphinx mais uma vez se mostrou bastante inferior aos demais, apresentando WER de 72,6% e xRT de 2,5, enquanto o Julius apresentou WER de 4,2% e xRT de 1,3, e o Google apresentou WER de 3,1% e xRT de 1,3. Isso os levou a concluir que a única vantagem do PocketSphinx é não depender de Internet, sem a qual reconhecimento em nuvem não é possível. Porém, havendo conexão disponível, tanto o Julius quanto o Google se mostraram melhores opções, com a diferença de que o Julius é *software* livre e permite impor gramáticas.

2.2 SISTEMAS DE DIÁLOGOS

Sistemas de diálogos são sistemas de computador capazes de conversar com um ser humano, sendo capazes de receber e processar informações deste e gerar informações dinâmicas para ele, seguindo o fluxo de uma conversa (ZUE; GLASS, 2000). Este tipo de interface entre homem e os sistemas computacionais surge naturalmente no contexto de crescimento da necessidade dos computadores na vida das pessoas, em especial com a explosão da Web, e também dos avanços em áreas de linguística computacional como a de processamento de linguagem natural, levando a interação homem-máquina o mais próximo possível da comunicação natural dos seres humanos (LESTER et al., 2004) (CLARK et al., 2012) (KAPETANIOS et al., 2013).

Os sistemas de diálogo podem ter diversas arquiteturas e utilizar as mais variadas formas de troca de informação: textos, voz, gráficos, gestos, entre outros. Sistemas de diálogos falados são aqueles que utilizam voz para comunicação, seja no sentido humano-computador (reconhecimento de voz) ou no sentido computador-humano (síntese de voz). Deste ponto em diante do trabalho, sempre que se utilizar o termo sistemas de diálogos será em relação a sistemas de diálogos falados.

Jurafsky e Martin (2008) discutem as características gerais de um sistema de diálogo. A principal delas é a presença do gerenciador do diálogo, que é o componente do sistema que irá cuidar do andamento do diálogo, mantendo o estado deste, suas transições e definindo a melhor estratégia para a continuação da conversa. Tipicamente, cada ciclo de execução do sistema de diálogo possui os seguintes passos (esquematizados na Figura 2.4):

1) O ser humano (usuário) fala algo ao sistema;
2) O sistema identifica as informações recebidas do usuário utilizando reconhecimento de voz;
3) As informações recebidas são tratadas, geralmente utilizando técnicas de processamento de linguagem natural;
4) Processadas as informações recebidas, o gerenciador do diálogo as analisa e movimenta a máquina de estados do diálogo, dando continuidade ao fluxo da conversa;
5) Decidido o próximo estado do diálogo, o sistema gera as informações que devem ser enviadas ao usuário, usando para isso técnicas como processamento de linguagem natural;
6) Por último, o sistema repassa as informações geradas ao usuário utilizando síntese de voz e o ciclo recomeça.

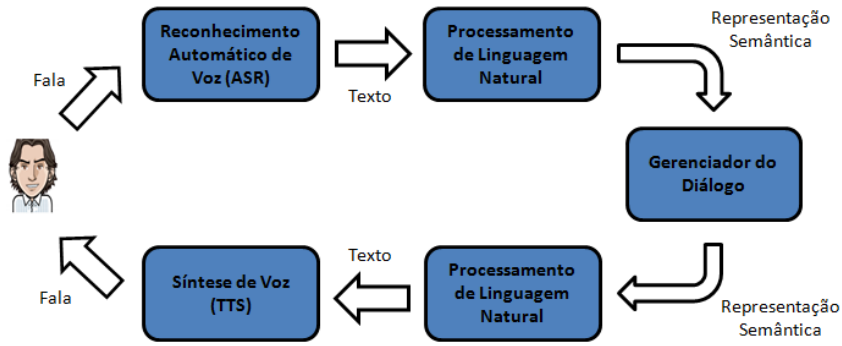


Figura 2.4 – Ciclo de execução típico de um sistema de diálogo. Adaptada de Traum (2012).

Outra característica importante de um sistema de diálogo é o tipo de interação com o usuário. Existem três tipos de interação: direcionada pelo sistema; direcionada pelo usuário; e direcionada de maneira mista (TRAUM, 2008).

No primeiro tipo, o fluxo da conversa é direcionado pelo sistema, isto é, o sistema conduz o diálogo restringindo o grau de liberdade da resposta do usuário e evitando que o usuário tome a iniciativa do diálogo. Por exemplo, se o sistema quisesse confirmar uma informação com o usuário, ele diria “Você confirma? Diga ‘sim’ ou ‘não’.”. Uma vantagem deste tipo de interação é a facilidade de implementação, além do fato de que ela se aplica muito bem quando há uma sequência bem definida para o diálogo, sem que as interações sejam fortemente dependentes umas das outras. Por outro lado, este tipo de interação impede que o usuário possa participar de maneira mais livre do diálogo.

No segundo tipo citado, quem controla o fluxo do diálogo é o usuário, escolhendo o caminho a seguir sem qualquer tipo de restrição imposta pelo sistema. Por exemplo, o sistema se dirigiria ao usuário dizendo “O que posso fazer por você?” e o usuário diria o que quisesse como, por exemplo, “Gostaria do número de telefone do João Miguel”. Uma vantagem deste tipo de interação é a praticidade para o usuário, que poderia interagir com o sistema de uma forma bem natural. Por outro lado, este tipo de interação torna a tarefa do gerenciador de diálogo extremamente complexa e de difícil implementação.

O último tipo de interação é uma mistura dos dois primeiros. Nele, o sistema busca controlar o fluxo do diálogo, porém sem restringir tanto as respostas que o usuário pode dar e, principalmente, sendo flexível para tentar tratar as iniciativas de mudança de fluxo oriundas do usuário. Por exemplo, neste tipo de interação, o sistema poderia dizer ao usuário “Diga o nome da pessoa cujos dados você deseja obter.”, esperando que o usuário informasse o nome da pessoa para, então, perguntar que informação é desejada. Porém, se a resposta do usuário fosse “Quero o número de fax da Laura Silva”, o sistema seria flexível para entender que houve uma mudança de fluxo do diálogo e responder ao usuário já com a informação pedida. Este tipo de interação busca incorporar as vantagens de ambos os tipos anteriores, porém incorpora também as desvantagens, de forma que o tipo de interação a ser utilizada depende muito das características que se busca para o sistema de diálogo.

Outro ponto de interesse no estudo dos sistemas de diálogos são os métodos de avaliação de tais tipos de sistemas. A avaliação é importante, pois representa uma forma de comparar diversos sistemas e de entender o quão útil cada um será para o usuário final.

Segundo Bernsen et al. (2007), a avaliação de um sistema de diálogo é complexa devido à natureza interativa de tais sistemas e também devido às diferenças de percepção de desempenho entre diferentes usuários. Por isso, é necessário avaliar não só os componentes individuais do sistema, como também seu desempenho global e o nível de percepção do usuário final.

Para a avaliação individual de cada componente de um sistema de diálogo, podem-se utilizar, como critérios de avaliação, métricas tradicionais como a taxa de erro por palavra para o reconhecedor de voz; taxa de erro de quadros semânticos para o módulo de processamento de linguagem natural; e grau de relevância da resposta selecionada para o gerenciador de diálogo.

Para uma avaliação do desempenho geral do sistema de diálogo, alguns critérios objetivos que podem ser utilizados são: taxa de sucesso do diálogo; número médio de ciclos de interação entre o sistema e o usuário em uma ligação; o número de repetições necessárias; a duração do diálogo; e o tempo de espera durante a ligação. Por último, para avaliar a percepção do usuário final, pode-se solicitar que respondam a questionários sobre o sistema.

Além de ser o componente principal de um sistema de diálogos, o gerenciador do diálogo é um componente bastante complexo e precisa lidar com diversos desafios. Vários são os inconvenientes que podem resultar em interações pouco eficientes com o usuário: erros de reconhecimento; fuga de contexto por parte do usuário; dificuldade de entendimento do usuário sobre o fluxo de diálogo; entre outros. Zue (2004) analisa diversas situações especiais para as quais o gerenciador de diálogos deve estar preparado. Por exemplo, o gerenciador deve ser capaz de resolver ambiguidades que possam existir na informação recebida do usuário (possivelmente conduzindo o fluxo do diálogo para um subdiálogo com uma pergunta de confirmação como “você disse gato ou pato?”); ou então tratar informações incompletas (possivelmente questionando o que falta como “e qual o número da casa?”); ou ainda manter a quantidade de informações prestadas ao

usuário em um nível inteligível (por exemplo, sugerindo filtros adicionais para uma listagem solicitada como “eu encontrei mais de vinte voos, você tem alguma preferência por companhia aérea?”). Além disso, o gerenciador de diálogo deve ser projetado de forma a ajudar o usuário a atingir seu objetivo, sendo preparado para sugerir fluxos que facilitem isso ou para oferecer alternativas quando aquilo que o usuário deseja não está disponível.

Por todo o exposto, fica claro que o projeto de sistemas de diálogo é uma tarefa que pode atingir um nível de complexidade bastante elevado. Vários são os trabalhos cujo tema é o projeto e avaliação deste tipo de sistema como os de Kamm e Walker (1997), Sadek (1999), Litman e Pan (2002), Sonntag et al. (2010) e Benyon et al. (2013). Além disso, vários são os trabalhos que buscam resolver problemas ou trazer melhorias aos gerenciadores de diálogo, como o trabalho de López-Cózar et al. (2011) que busca melhorar a detecção de emoção do usuário em sua fala, trazendo com isso mais subsídios para o trabalho do gerenciador de diálogos; o trabalho de Bohus e Rudnicky (2005) que busca melhorar a forma de avaliar o grau de confiança do sistema nas informações que foram recebidas do usuário; e o trabalho de Wang e Swegles (2013) que busca endereçar o problema de desambiguizar a informação passada pelo usuário para o computador.

Sistemas de diálogos estão presentes, hoje, em diversas áreas do conhecimento como educação, negócios, saúde e entretenimento. As aplicações são muitas como: serviços de informação a clientes, *help desk*, suporte técnico, guia de compras ou navegação em sítios Web, diagnósticos, educação e treinamentos, entre outros (LESTER et al., 2004). Em especial, sistemas de diálogos são amplamente utilizados nos *call centers* de empresas, como forma de tornar seu atendimento mais robusto e eficiente e ajudar a diminuir a sobrecarga de trabalho dos operadores. Para tal tipo de aplicação, os sistemas de diálogos são comumente conhecidos como sistemas IVR (*Interactive Voice Response* ou Unidade de Resposta Audível).

2.2.1 *Call Center*

Call center é um termo genérico para designar um escritório centralizado de uma empresa responsável por receber e transmitir uma grande quantidade de requisições telefônicas, respondendo por diversos serviços como vendas, informações, suporte, ouvidoria, entre outros, disponibilizados pela empresa para os clientes. E justamente por ser esse canal direto com o cliente, os *call centers* têm sido importante foco de investimento para as empresas (SUBRAMANIAM, 2008).

Existem basicamente dois tipos de chamadas telefônicas em um *call center*: as de entrada, realizadas pelos clientes que buscam obter serviços, informações ou um canal de comunicação com a empresa; e as chamadas de saída, feitas por agentes das empresas diretamente para os clientes, buscando oferecer produtos, serviços ou vantagens de maneira direcionada e individualizada. As chamadas de interesse para este trabalho são as de entrada.

A importância de um *call center* para os negócios de uma empresa é tema de diversos estudos como os de Aksin et al. (2007), que analisa com detalhes os desafios de manutenção de uma central deste tipo, e o de As-Saber e Hossain (2008), que discute as vantagens de *call centers* com aplicações governamentais em países emergentes.

O número de chamadas de clientes em *call centers* é elevado e é muito importante atender a todas com rapidez, qualidade e efetividade. Para utilizar atendentes humanos para todas as chamadas, seria preciso contar com uma equipe enorme, o que, mesmo que possível, levaria a uma não uniformidade no atendimento. Por outro lado, manter as ligações dos clientes em filas, aguardando sua vez de ser atendido, não é considerada uma boa prática e pode causar grande descontentamento. É por essa razão que as empresas buscam soluções em *software* capazes de realizar um primeiro atendimento aos clientes, coletando as informações básicas destes e direcionando-os para os atendentes corretos, isso quando não é possível resolver diretamente o seu problema. Abdullateef et al. (2011) abordam a importância de resolver a requisição do cliente em sua primeira ligação como fator determinante para sua satisfação com os serviços da central.

Os *softwares* que realizam este primeiro atendimento são sistemas de diálogos conhecidos como IVR. Tradicionalmente, eles funcionam por meio de menus, onde é fornecida ao usuário uma série de opções associadas a números e este deve escolher a opção desejada discando o número correspondente. Por exemplo, se em um determinado atendimento é necessário saber se o usuário quer falar com o setor de recursos humanos ou de suporte, o sistema pode falar: “Para falar com o setor de recursos humanos digite 1; e para falar com suporte digite 2.”. O usuário deve então discar o número correspondente da sua seleção.

Este tipo de menu possui limitações e nem sempre pode ser aplicado. Por exemplo, ao tratar de um serviço de venda de passagens aéreas, o sistema não tem como listar todos os possíveis destinos e mapeá-los para números a serem discados. Neste caso, é preciso utilizar recursos adicionais de sistemas de diálogos, como o reconhecimento de voz: basta que o cliente diga o destino e o sistema saberá tratar do pedido, tornando a conversa muito mais natural, agradável ao cliente e eficiente. E quanto maior for a capacidade de processamento de linguagem natural deste sistema de diálogo, maior será sua eficiência e melhor serão os resultados para a empresa, assim como maior será a satisfação do cliente (BENNINGTON et al., 2000).

Por último, convém lembrar que, obviamente, custos também possuem impacto e importância nas decisões das empresas em relação à implantação de um *call center*, como discutido por Bolte e Fleischman (2007). Para a maioria das empresas, projetar seu próprio sistema de diálogo, mesmo que simples, não é factível, o que as leva a procurar terceiros que possam realizar o serviço. Entretanto, os custos para tal não são baixos, o que pode dificultar, principalmente para pequenas empresas, manter um *call center* em funcionamento.

2.2.2 Arquiteturas para Sistemas de Diálogos

Para implementar um sistema de diálogo, existem várias arquiteturas e abordagens possíveis e cada uma delas, possivelmente, pode se traduzir em um

framework que permite que novos sistemas sejam produzidos de maneira mais rápida e efetiva, utilizando tal arquitetura.

O programa Eliza foi um marco de referência na história dos sistemas de diálogos. Ele foi construído para participar de diálogos de psicoterapias. Usando estrutura local com memória limitada, o programa Eliza buscava utilizar reconhecimento de padrões, regras de transformações e detecção de palavras-chave para processar as informações de entrada do usuário e gerar suas saídas (WEIZENBAUM, 1966).

Com o sucesso de Eliza, Richard Wallace criou a Artificial Intelligence Markup Language (AIML), uma linguagem compatível com o XML destinada a criar agentes virtuais de bate-papo (*chat-bots*). Esta linguagem se baseia na interação por estímulo e resposta e engloba as características do programa Eliza, trazendo funcionalidades adicionais. Ela acabou sendo incorporada no *framework* A.L.I.C.E[11], acrônimo de *Artificial Linguistic Internet Computer Entity*.

Sistemas de diálogos também podem se servir de *scripts* para configurar as ações a serem tomadas durante a conversação; máquinas de estados e árvores de decisão para gerenciar o fluxo da conversa; estratégias estatísticas ou de gramáticas que restrinjam o diálogo; assim como de variações de tais técnicas (TRAUM, 2012). Um exemplo de *framework* que busca agrupar diversas dessas abordagens é o Olympus (BOHUS et al., 2007).

Seguindo a ideia do AIML, o *World Wide Web Consortium* (W3C) padronizou uma linguagem compatível com XML, chamada VoiceXML[12], a ser utilizada para aplicações de voz. A ideia do VoiceXML é seguir os moldes do HTML, porém através de um *Voice Browser*, em vez de um navegador de Internet. O fluxo é análogo: primeiro o usuário solicita informação do servidor através do *Voice Browser*; este encaminha a requisição ao servidor que a processa e gera o VoiceXML de resposta, repassando-o novamente ao *Voice Browser*, que irá renderizar a resposta ao cliente para que seja iniciada uma nova interação. Tsai (2005) apresenta um sistema de diálogo utilizando VoiceXML.

---

[11] http://www.alicebot.org/aiml.html. Acesso em: 21 nov. 2013.
[12] http://www.w3.org/TR/voicexml21. Acesso em: 21 nov. 2013.

Para o cenário específico de uma central telefônica do tipo PABX, o *software* livre Asterisk propõe uma arquitetura para gerenciar diálogos baseada em *scripts* extensíveis, onde novas funcionalidades, através de módulos escritos em qualquer linguagem, podem ser adicionadas no fluxo do diálogo, tornando possível o processamento de linguagem natural. Goel e Bhattacharya (2010) apresentam uma visão teórica e de implementação de um sistema de diálogos no Asterisk.

Vale lembrar que sempre é possível combinar diferentes tipos de arquiteturas, em uma tentativa de aproveitar os pontos positivos de cada uma. O grupo L2F[13], de Lisboa, possui uma solução para sistemas de diálogos baseada em Asterisk e VoiceXML.

2.3 ASTERISK

O Asterisk[14] é um *software* livre que pode ser utilizado para diversas aplicações, principalmente em telefonia. O Asterisk é especialmente preparado para ser usado como comutador de ramais privados (PBX) sobre IP, *gateway* de voz sobre IP (VoIP), unidade de resposta audível (IVR) e distribuidor de chamadas. Segundo a empresa Digium[15], que mantém o Asterisk, ele é atualmente usado em mais de um milhão de sistemas em mais de 170 países.

O Asterisk deve ser utilizado em sistema operacional Linux e, por possuir uma arquitetura fortemente modular, é facilmente adaptável. Virtualmente, este pode se comunicar com qualquer *hardware*, linguagem de programação ou protocolo. Um dos ideais dos desenvolvedores do Asterisk é que o mesmo possa, no futuro, atender a todos os protocolos existentes.

É sabido que o Asterisk pode se comunicar com a rede pública de telefonia comutada (PSTN), que é, basicamente, a rede de telefonia usualmente utilizada,

---

[13] https://tecnovoz.l2f.inesc-id.pt. Acesso em: 21 nov. 2013.
[14] http://www.digium.com/en/products/asterisk. Acesso em: 21 nov. 2013.
[15] http://www.digium.com. Acesso em: 21 nov. 2013.

composta por linhas telefônicas convencionais, cabos de fibra óptica, rede de celulares, satélites de comunicação, dentre outros. O Asterisk pode também trabalhar como um servidor VoIP, principalmente usando o protocolo de iniciação de sessão (SIP), que possibilita, dentre outras coisas, a comunicação entre *softphones*, além de ser bastante utilizado por serviços VoIP que se conectam a PSTN, como o Skype[16].

2.3.1 Configuração e Arquitetura do Asterisk

O Asterisk pode ser visto como um sistema com duas partes: um núcleo e um conjunto de APIs de módulos carregáveis para abstração. O núcleo do Asterisk contém os módulos que lidam com o desempenho do sistema, o gerenciamento de chamadas, *codecs* de entrada e saída, entre outros. Já o conjunto de APIs é composto de quatro API definidas para módulos carregáveis, facilitando a abstração de *hardware* e protocolos. As quatro APIs são: API de Aplicações (*Application API*), API de Canais (*Channel API*), API de Formato de Arquivos (*File Format API*) e a API de Transcodificador (*Transcoder API*). A Figura 2.5 ilustra a macro-arquitetura do Asterisk.

[16] http://www.skype.com. Acesso em: 21 nov. 2013.

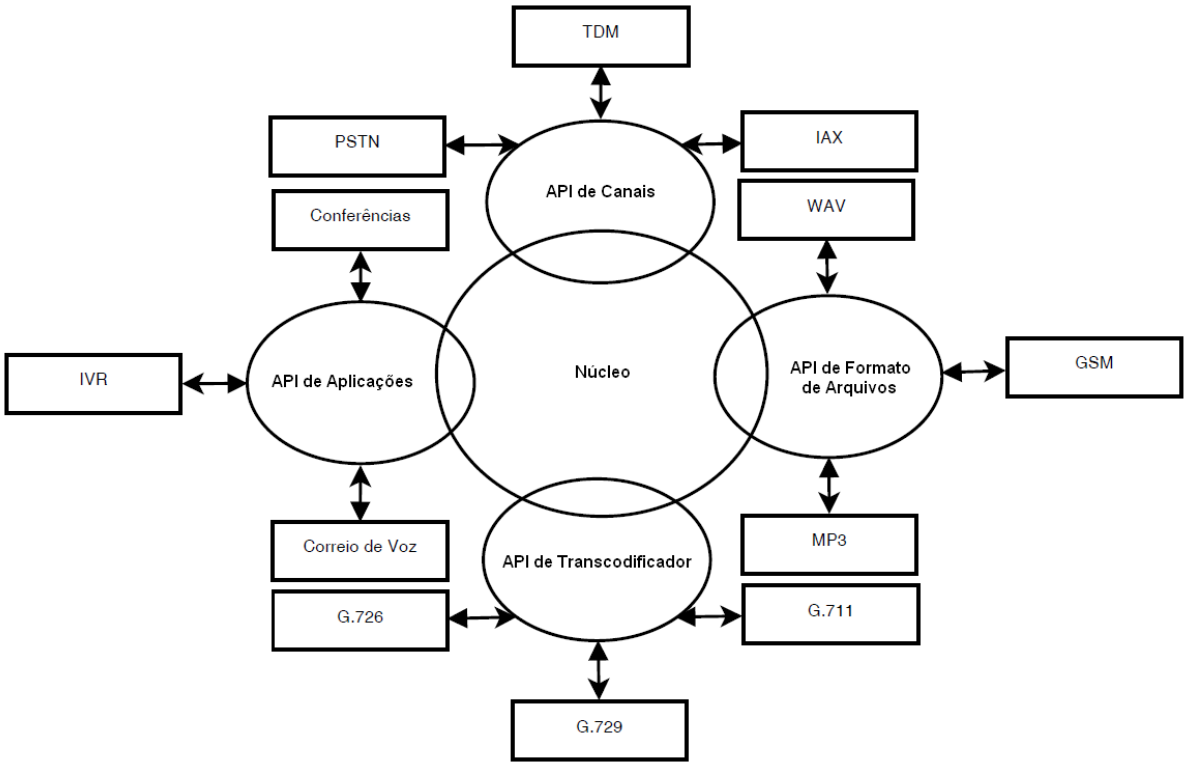


Figura 2.5 – Arquitetura de módulos carregáveis do Asterisk.

Essa arquitetura modular permite que a integração do Asterisk com novas tecnologias seja implementada sem muito esforço. Por exemplo, caso um novo protocolo ou *hardware* seja inventado, basta escrever um módulo da *Channel API* para que o Asterisk possa se comunicar com essa nova tecnologia.

O Asterisk é controlado por uma série de arquivos de configuração situados em diversos diretórios do Linux. Exemplos de configurações e recursos são: gravações de caixa postal, *prompts* de voz e arquivos de configuração de transferência e fila de chamadas, dentre outros. A seguir, são descritos alguns desses diretórios de configuração (assumindo-se a configuração padrão para o Asterisk).

1) **`/etc/asterisk`**: contém os arquivos de configuração e de lógica do sistema;
2) **`/usr/lib/asterisk/modules`**: contém os módulos carregáveis do Asterisk, que controlam várias de suas funcionalidades;

3) **`/var/lib/asterisk/sounds`**: contém os arquivos de sons usados no Asterisk, como *prompts* de voz, músicas para chamada em espera, dentre outros;
4) **`/var/lib/asterisk/agi-bin`**: contém os *scripts Asterisk Gateway Interface* (AGI) e *Extended Asterisk Gateway Interface* (EAGI).

2.3.2 Unidade de Resposta Audível

Ao chegar ao Asterisk, uma chamada é direcionada a um contexto, cujas ações a serem tomadas estão especificadas, sob forma de *script*, em um arquivo chamado *extensions.conf*. A linguagem de *script* utilizada se chama *dialplan*. Esse *script* relativo ao contexto é que especifica o que o sistema deve fazer com a chamada. Exemplos de comandos possíveis são: ligar e conectar outro usuário, atender à ligação, tocar uma música, esperar que o usuário digite um número, desligar, dentre outros.

A chamada pode ser direcionada para diferentes contextos dependendo de sua origem. Um arquivo de configuração específico para a placa utilizada é responsável por suas configurações, incluindo o contexto que deve ser utilizado pelo Asterisk ao receber a chamada. Por exemplo, para a placa DigiVoice[17] VB0408PCI, existiria o arquivo *digivoice.conf*. Neste arquivo, a parte que especificaria o contexto a ser utilizado para esta chamada está mostrado a seguir:

```
[port_config]
signalling=fxo
context=from-pstn
language=pt_BR
ports=>3
```

Neste exemplo, a chamada seria direcionada para o contexto **from-pstn**, cujo *script* de execução se encontra no arquivo *extensions.conf*.

[17] http://www.digivoice.com.br. Acesso em: 21 nov. 2013.

2.3.3 Controle da Ligação com o Dialplan

O *script* que faz o controle da ligação, isto é, define as ações a serem tomadas durante o processamento da ligação, é uma das partes mais importantes do Asterisk. Ele comumente é conhecido pelo mesmo nome da linguagem em que é escrito, ou seja, *dialplan*. Esta linguagem de script é bastante adaptável, podendo, facilmente, proporcionar a interação do Asterisk com sistemas externos.

O *dialplan* contém quatro conceitos: contextos, extensões, prioridades e aplicações. Cada conceito será brevemente discutido a seguir.

2.3.3.1 Contextos

Um *dialplan* é separado em seções chamadas de contextos. Dentro de um contexto as extensões estão isoladas, isto é, se em um contexto o usuário digitar 0 significar “confirma”, em outro contexto o digito 0 pode significar “não confirma”. Isso é configurável e o contexto para qual o usuário deve ser direcionado é definido de acordo com sua origem, como já discutido na seção 2.3.2.

Contextos são definidos com seu nome entre colchetes. Como exemplo, a seguir é definido o contexto “exemplo”.

```
[exemplo]
```

Todas as instruções contidas depois da definição de um contexto pertencerão a este, até que um novo contexto seja definido. Existem dois contextos reservados que são o *general* e o *globals*. O primeiro lida com algumas configurações e o segundo define variáveis visíveis simultaneamente a todos os contextos.

2.3.3.2 Extensões

Em aplicações de telefonia, o termo extensão comumente se refere a um identificador numérico que, quando discado, realiza uma ligação a um telefone. No Asterisk, uma extensão é algo mais poderoso que define, de maneira única, o passo-a-passo de ações a serem tomadas para a ligação.

Dentro de um contexto, é possível definir tantas extensões quantas forem necessárias. Quando uma extensão for ativada (seja por uma chamada recebida ou por dígitos discados em um canal), o Asterisk irá seguir os passos definidos no *dialplan* para essa extensão.

A sintaxe para uma extensão é a palavra ***exten*** seguida por uma seta formada pelo sinal de igual (=) e o sinal de maior (>), seguida pelo número (ou nome) da extensão, sua prioridade e, finalmente, a aplicação (ou comando) que deve ser executado, separados por vírgula como mostrado a seguir.

```
exten => nome,prioridade,aplicação()
```

No exemplo a seguir é mostrada uma aplicação no contexto “exemplo1” que atende a uma ligação para o número 12. No exemplo, o nome da extensão é 12, a prioridade é 1 e a aplicação é ***Answer()***.

```
[exemplo1]
exten => 12,1,Answer()
```

2.3.3.3 Prioridades

Cada extensão pode conter múltiplos passos, chamados prioridades. As prioridades são numeradas e executadas sequencialmente começando de 1, e cada uma delas executa uma determinada aplicação. Como exemplo, a extensão a seguir atende a ligação para o número 12 (prioridade 1), espera 5 segundos (prioridade 2) e então a desliga (prioridade 3).

```
[exemplo2]
exten => 12,1,Answer()
exten => 12,2,Wait(5)
exten => 12,3,Hangup()
```

2.3.3.4 Aplicações

As aplicações são as ações que as extensões devem executar. Cada aplicação faz uma determinada ação no canal aberto, como, por exemplo, tocar um som; receber um número discado; consultar um banco de dados; discar para outro canal; e desligar a ligação, dentre outras.

Nos exemplos das seções 2.3.3.2 e 2.3.3.3 foram apresentadas as aplicações ***Answer()***, ***Wait()*** e ***Hangup()***, responsáveis por atender a ligação, esperar por um tempo determinado, e desligar a ligação, respectivamente. Algumas aplicações, como a ***Answer()***, não precisam de argumentos, porém para outras, como a ***Wait()***, são necessários um ou mais argumentos, que devem estar separados por vírgula.

Através das aplicações, o Asterisk pode se conectar a programas, escritos em qualquer linguagem de programação, que escrevam e leiam em sua entrada e saída padrão, respectivamente. Dessa forma, durante uma ligação, qualquer programa pode ser executado pelo Asterisk. O que possibilita esse tipo de operação é a interface denominada *Asterisk Gateway Interface* (AGI).

A AGI é projetada para enviar um comando para o Asterisk, escrevendo em sua saída padrão; e para receber as respostas através de sua entrada padrão. Para executar um programa AGI, a aplicação ***AGI()*** deve ser chamada da seguinte forma:

```
AGI(command[,arg1[,arg2[,...]]])
```

Nesta chamada, ***command*** é o programa a ser executado e ***arg1***, ***arg2***, sucessivamente, são os argumentos que este necessita. Tais argumentos são passados ao programa como se fossem argumentos de linha de comando. Inicialmente, além de receber os argumentos passados pela aplicação ***AGI()***, uma

série de variáveis de ambiente são escritas na entrada padrão do programa AGI, no seguinte formato:

<<nome_variável_agi>>: <<valor>>

Dentre tais variáveis, está o número do usuário que originou a chamada (variável ***agi_callerid***) e o contexto atual (variável ***agi_context***).

A seguir é mostrado um programa AGI escrito na linguagem Python (chamado *say_wait.py*) que quando executado espera por ***arg1*** segundos, fala os caracteres ***arg2*** e retorna.

```
1    import sys, time
2
3    env_vars = dict()
4    while True:
5         line = sys.stdin.readline()
6         if line == '\n':
7              break
8         else:
9              var, _, value = line.partition(':')
10             env_vars[var] = value
11
12   time.sleep(sys.argv[1])
13
14   sys.stdout.write('SAY ALPHA "%s"\n' % sys.argv[2])
15   sys.stdout.flush()
16   status, _, result = sys.stdin.readline().partition('=')
17   if status != '200 result' or result != 0:
18        sys.stderr('Erro no comando SAY ALPHA\n')
19        sys.exit(result)
```

Nas linhas de 3 a 10, são lidas as variáveis de ambiente. Na linha 12, a execução fica em espera por ***arg1*** segundos. Nas linhas 14 a 16, é mostrado como o programa AGI consegue interagir com o Asterisk: na linha 14, ele escreve em sua saída padrão o comando a ser lido e executado pelo Asterisk; enquanto na linha 15 se garante que o comando não fique em um *buffer* parado; e na linha 16, o programa lê o resultado da execução, gravado pelo Asterisk em sua entrada padrão. O comando, neste caso, é o ***SAY ALPHA*** que falará cada um dos caracteres

passados como parâmetro. Após o Asterisk falar todos os caracteres, ele escreverá na entrada do programa AGI o retorno do comando, cujo formato padrão é:

<code> result=<result> [data]

Neste formato, ***<code>*** é o código de retorno da comunicação com o Asterisk (“200” representa sucesso, enquanto “5xx” significa erro); ***<result>*** é o retorno do comando (comumente, “-1” significa erro, enquanto “0” representa sucesso); e ***[data]*** é utilizado opcionalmente por alguns comandos para passar informações adicionais ao programa AGI.

Por último, o programa AGI processa o retorno do comando executado pelo Asterisk nas linhas 17 a 19.

Uma variação da AGI é a *Extended* AGI (EAGI) que, além de todos os recursos disponibilizados pela AGI, ainda possibilita o acesso ao canal de áudio pelo descritor de arquivo “3”. Para exemplificar, abaixo é mostrado um programa (chamado *record.py*) que grava 5 segundos de áudio.

```
1    import sys, time
2
3    env_vars = dict()
4    while True:
5         line = sys.stdin.readline()
6         if line == '\n':
7              break
8         else:
9              var, _, value = line.partition(':')
10             env_vars[var] = value
11
12   channel = os.fdopen(3, 'r')
13   with open('/tmp/file.raw', 'w') as out_file:
14        out_file.write(channel.read(2 * 8000 * 5))
```

Mais uma vez, nas linhas de 3 a 10, são lidas as variáveis de ambiente. Em seguida, na linha 12, o canal é aberto, para leitura, através do descritor de arquivo “3”. O arquivo ***“/tmp/file.raw”*** é criado na linha 13 e, finalmente, na linha 14, são lidos 80.000 *bytes*, já que há 2 *bytes* por amostra, cada segundo contém 8.000

amostras e se quer gravar 5 segundos. As amostras são inteiros de 16 *bits* com sinal e o som possui 1 canal.

O *dialplan* mostrado a seguir exemplifica a utilização de programas AGI e EAGI. Nele, o Asterisk deve atender a ligação, aguardar 2 segundos, falar os caracteres “ABC”, gravar 5 segundos da ligação e, então, desligar.

```
[exemplo3]
exten => 12,1,Answer()
exten => 12,2,AGI(say_wait.py,2,ABC)
exten => 12,3,EAGI(record.py)
exten => 12,4,Hangup()
```

Existem vários comandos AGI para interação com o Asterisk e também diversas aplicações disponíveis para uso no *dialplan*. A seguir, algumas dessas aplicações são descritas.

`Answer()`: atende uma ligação;

`Playback(arquivo_de_audio)`: toca o áudio do parâmetro “arquivo_de_audio”;

`Set(var=valor)`: atribui o valor do parâmetro “valor” à variável do parâmetro “var”;

`Goto(contexto,extensão,prioridade)`: direciona a chamada para o contexto, extensão e prioridade especificados nos parâmetros homônimos;

`GotoIf(expressão?destino1:destino2)`: se a expressão “expressão” for verdadeira, a chamada é direcionada para o destino “destino1”; se ela for falsa, a chamada é direcionada para o destino “destino2”. Qualquer um dos destinos “destino1” e “destino2” é formado pela tupla: contexto, extensão e prioridade, sendo que o contexto e/ou a extensão podem ser omitidos, sendo assumido o estado atual;

`Hangup()`: desliga a ligação;

`AGI(command[,arg1[,arg2[,...]]])`: executa o programa AGI chamado “command”, que recebe como argumentos os *n* parâmetros passados para a aplicação ***AGI()*** depois de “command”;

`EAGI(command[,arg1[,arg2[,...]]])`: executa o programa EAGI chamado “command”, que recebe como argumentos os *n* parâmetros passados para a aplicação ***EAGI()*** depois de “command”.

2.3.4 Reconhecendo Voz no Asterisk

Para possibilitar o uso de reconhecimento de voz em um *dialplan* no Asterisk, Batista (2013) desenvolveu um programa EAGI, na linguagem Phyton, que utiliza o software Coruja (SILVA et al., 2010) para reconhecer fala restrita a uma determinada gramática e retornar o resultado em variáveis de contexto capazes de serem lidas pelo Asterisk.

Duas ferramentas auxiliares também foram desenvolvidas: um *script* e um *plugin*. O primeiro, escrito em Phyton e chamado *pysapixml2julius*, é um conversor de gramáticas do formato SAPI XML[18] para o formato utilizado pelo decodificador Julius, que é usado pelo Coruja. O conversor possui suporte às *tags* GRAMMAR, RULE, PHRASE, LIST e OPT e conta com uma *tag* adicional chamada RETURN que torna possível o retorno de uma *string* arbitrária para uma dada regra. Já o *plugin* permite que o decodificador Julius utilize como entrada de áudio o descritor de arquivo “3”, que é onde o Asterisk disponibiliza áudio da ligação, como explicado na seção 2.3.3.4.

O programa EAGI desenvolvido para a integração do Coruja com o Asterisk é chamado *adintool.py* e deve ser executado da seguinte forma:

`EAGI(adintool.py, arquivo_jconf, prefixo_gramática, tempo_máximo, número_tentativas, mensagem_nova_tentativa, arquivo_áudio)`

O parâmetro “arquivo_jconf” é o arquivo de configuração do Julius e o parâmetro “prefixo_gramática” é prefixo dos arquivos de gramática do Julius (**.dfa**,

[18] http://msdn.microsoft.com/en-us/ms723632(VS.85).aspx. Acesso em: 23 nov. 2013.

.dict, **.voca** e **.ret**). O parâmetro “tempo_máximo” representa o tempo máximo, em segundos, em que o reconhecimento deve ocorrer, enquanto o parâmetro “número_tentativas” representa quantas vezes o reconhecedor deve tentar identificar o que está sendo dito. O parâmetro “mensagem_nova_tentativa” deve trazer uma mensagem a ser informada ao usuário caso uma nova tentativa de reconhecimento deva ser tentada. Por último, o parâmetro “arquivo_áudio” indica o arquivo em que o áudio recebido pelo reconhecedor será gravado. Ao ser executado, o programa EAGI inicia a execução do Coruja, realiza o reconhecimento de voz e devolve as seguintes informações: resultado do reconhecimento, retorno definido pela regra gramatical em XML, confiança do reconhecimento por palavra e confiança do reconhecimento. Tais informações são devolvidas, respectivamente, nas variáveis *RECRESULT*, *RECRET*, *RECCONFIDENCEPERWORD* e *RECCONFIDENCE*.

2.3.4.1 Exemplo de Utilização do Coruja no Asterisk

Neste exemplo, ao atender a ligação, o Asterisk deve prestar uma informação qualquer ao usuário, solicitando que este confirme se a entendeu ou não. Em caso positivo, o sistema encerra a chamada. Em caso negativo, o sistema encaminha a chamada para um atendente.

Para reconhecer a confirmação ou a negação do usuário, a seguinte gramática foi proposta por Batista (2013):

```
<GRAMMAR>
<RULE ID="1" TOPLEVEL="ACTIVE">
      <LIST>
            <PHRASE> isso <OPT> mesmo </OPT></PHRASE>
            <PHRASE> positivo </PHRASE>
            <PHRASE> correto </PHRASE>
            <PHRASE> confirmo </PHRASE>
            <PHRASE> confirmado </PHRASE>
            <PHRASE> confirma </PHRASE>
            <PHRASE> certo </PHRASE>
            <PHRASE> sim
                  <OPT> sim </OPT>
```

```
                  <OPT>
                        <LIST>
                              <PHRASE> confirmo </PHRASE>
                              <PHRASE> confirmado </PHRASE>
                              <PHRASE> confirma </PHRASE>
                        </LIST>
                  </OPT>
                  <OPT> sim </OPT>
            </PHRASE>
      </LIST>
      <RETURN> 199998 1008 1 </RETURN>
</RULE>
<RULE ID="2" TOPLEVEL="ACTIVE">
      <LIST>
            <PHRASE> não <OPT> confirmo </OPT> <OPT> não </OPT> </PHRASE>
            <PHRASE> negativo </PHRASE>
            <PHRASE> incorreto </PHRASE>
            <PHRASE> errado </PHRASE>
      </LIST>
      <RETURN> 199999 1008 0 </RETURN>
</RULE>
</GRAMMAR>
```

Esta gramática possibilita que o usuário fale frases de confirmação como “sim confirmo”, “confirmado”, “sim confirma sim” e “sim sim”, dentre outras, como ilustrado na Figura 2.6. Para negar, o usuário pode falar frases como “não” e “incorreto”, dentre outras, como ilustrado na Figura 2.7. É importante notar que a regra de confirmação possui um retorno único para qualquer que seja a frase dita pelo usuário, assim como a regra de negação.

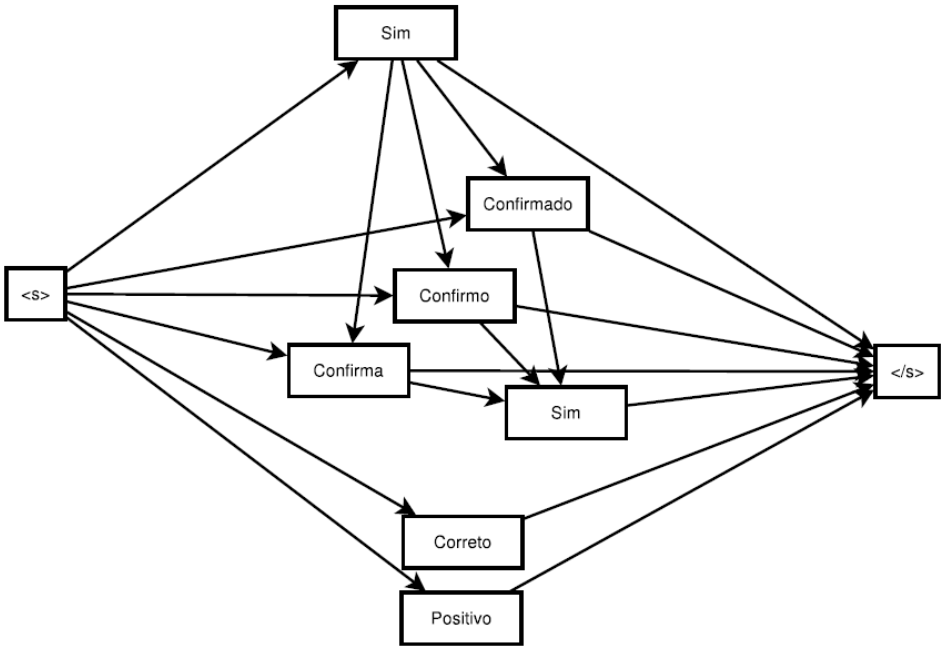


Figura 2.6 – Diagrama para regra de confirmação da gramática. Extraída de Batista (2013).

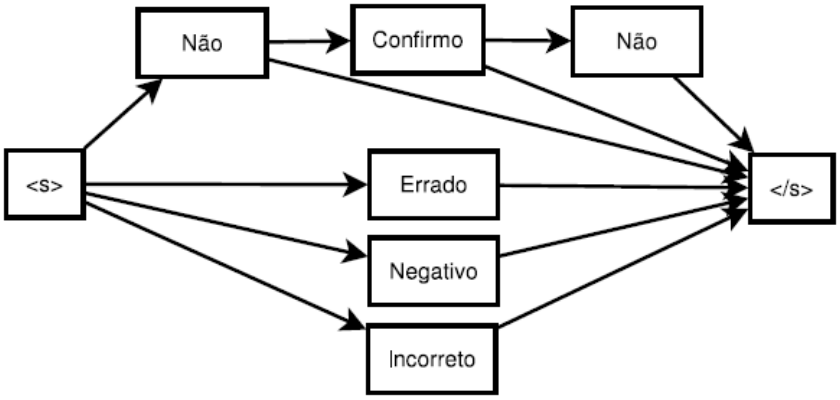


Figura 2.7 – Diagrama para regra de negação da gramática. Extraída de Batista (2013).

Uma vez gerado o arquivo XML da gramática, nomeado, por exemplo, ***sim_nao.xml***, o conversor *pysapixml2julius* deve ser executado, compilando a gramática e gerando os demais arquivos necessários ao decodificador Julius.

Por último, o *dialplan* é escrito de acordo com as especificações do exemplo, executando um *prompt* de áudio, iniciando o reconhecedor, aguardando a confirmação ou negação do usuário e direcionando a chamada para um atendente, em caso de negação. Segue:

```
[exemplo]
exten => 12,1,Answer()
exten => 12,2,Playback(solicita_anuencia)
exten => 12,3,EAGI(adintool.py,julius.jconf,sim_nao,6,1,,
                   /tmp/audio.wav)
exten => 12,4,Set(CONF=${RECRET:12:1})
exten => 12,5,GotoIf($[${CONF} = 1]?7:6)
exten => 12,6,Goto(atendente,1,1)
exten => 12,7,Hangup()
```

Como se pode observar, a prioridade 2 toca um arquivo de áudio que deve solicitar a anuência do usuário; em seguida, a prioridade 3 faz o reconhecimento da resposta do usuário; o resultado do reconhecimento é utilizado na prioridade 4, onde é atribuído à variável CONF o 13° caractere do reto rno da regra (“1” para a regra de confirmação e “0” para a regra de negação); na prioridade 5, é testado se o usuário respondeu confirmando ou negando: caso tenha havido confirmação, a ligação é encerrada na prioridade 7; caso contrário, a ligação é direcionada para um atendente (assumindo que existe a extensão atendente com este nome e prioridade 1) e somente quando a interação com o atendente acabar é que a ligação será encerrada na prioridade 7.

2.3.4.2 Testes de Validação do Reconhecedor Coruja no Asterisk

Batista (2013) realizou testes de validação do reconhecedor Coruja no Asterisk. Vários testes foram realizados usando uma gramática contendo o nome de 76 cidades, em três abordagens diferentes: com *softphone*, com telefone fixo VoIP e com celular. Todos os testes foram realizados em ambiente não controlado – um escritório –, onde existem variados ruídos como pessoas conversando, ar-condicionado, entre outros.

Os resultados obtidos foram satisfatórios para uso em unidades de resposta audíveis no Asterisk. O gráfico da Figura 2.8 mostra o resultado de um dos testes com *softphone*. Nele, um usuário com voz masculina utilizou um *headset* comum para repetir três vezes cada uma das 76 cidades da gramática.

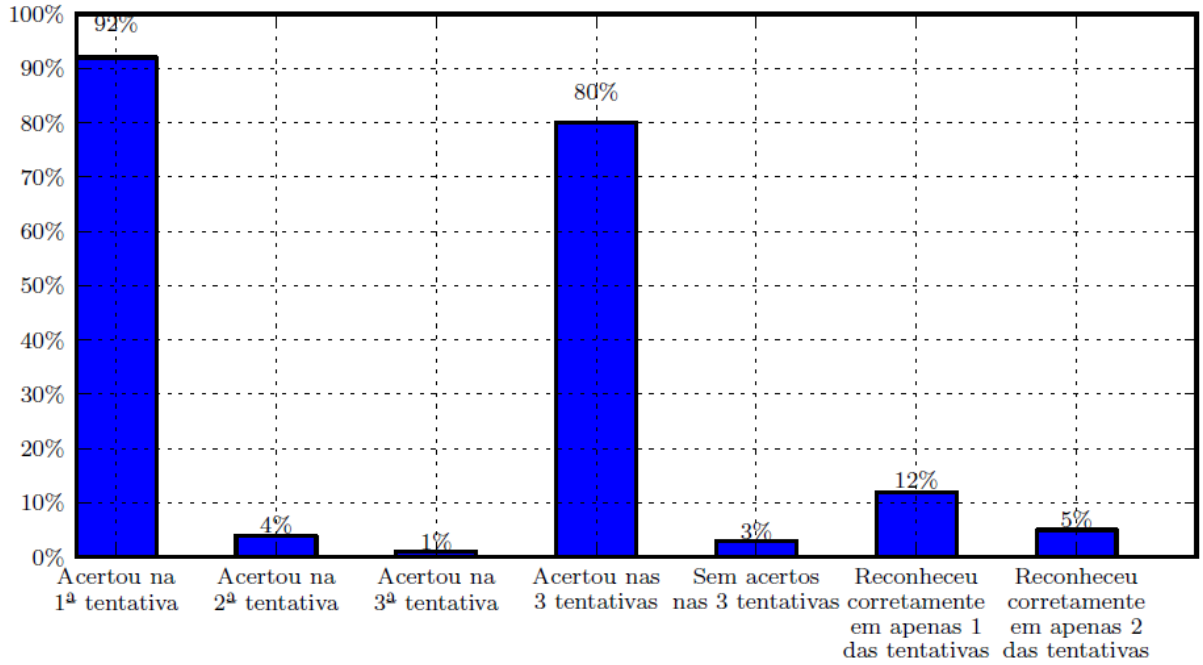


Figura 2.8 – Resultado do teste utilizando *softphone* com voz masculina. Extraída de Batista (2013).

Como se pode observar, em 92% dos casos, o reconhecedor teve sucesso em identificar a cidade falada na primeira tentativa. E em 80% dos casos, o reconhecedor acertou a cidade falada nas três tentativas.

3 APLICATIVO DIALOGBUILDER

3.1 CONSIDERAÇÕES INICIAIS

Como discutido na seção 2.2.1, *call centers* possuem grande importância para os negócios de uma empresa, porém os custos de instalação e manutenção deste tipo de central não são baixos.

O *software* livre Asterisk, discutido na seção 2.3, é capaz de prover para uma empresa um meio gratuito para implementação e hospedagem de um *call center*, bastando para isso que a empresa seja capaz de desenvolver suas próprias unidades de resposta audível, através da codificação de seus próprios *scripts dialplan*, de acordo com suas necessidades.

Entretanto, apesar de contar com vários recursos, como discutido na seção 2.3.3, a linguagem *dialplan* não é amigável, sendo difícil para um usuário sem experiência em programação entendê-la e conseguir gerar seus próprios *scripts*.

É neste contexto que o aplicativo DialogBuilder foi desenvolvido. Seu objetivo é prover ao usuário leigo uma forma intuitiva e simples para projetar seus sistemas de diálogos e gerar os respectivos *scripts dialplan* a serem utilizados no Asterisk.

Vale ressaltar que, como discutido na seção 2.2.2, existem várias arquiteturas possíveis para um sistema de diálogos, porém, a escolha de gerar *scripts dialplan* para Asterisk no DialogBuilder foi motivada pelo fato de que o Asterisk é um *software* livre propício para implantação de *call centers* e também pelo fato de que os *scripts dialplan* já são ferramentas nativas do Asterisk para construção de unidades de resposta audíveis, não sendo necessário instalar nada mais. Utilizar outra plataforma para sistemas de diálogos como o Olympus implicaria em que a empresa precisasse de uma solução adicional para a infraestrutura de *call center*; enquanto utilizar outra tecnologia, como o padrão VoiceXML, junto com Asterisk implicaria à empresa manter infraestrutura adicional para tal tecnologia, como um servidor extra para responder às requisições VoiceXML.

Por último, é importante ressaltar que existem soluções comerciais semelhantes ao DialogBuilder, como o Visual Dialplan[19]. Porém, além do DialogBuilder ser de código aberto, ele ainda traz o diferencial de um *wizard* em forma de diálogo que facilita ainda mais para que um usuário leigo possa ser capaz de projetar seus próprios sistemas de diálogo. Este *wizard* será discutido com mais detalhes na seção 3.3.5.2.

3.2 CARACTERÍSTICAS TÉCNICAS

O DialogBuilder é um aplicativo *desktop*, escrito na linguagem de programação Java[20], através do *framework* Swing[21]. Para seu desenvolvimento, utilizou-se a IDE NetBeans[22].

Para repositório de informações, utilizou-se base de dados H2[23] embutida na aplicação. E para a persistência de dados no H2 utilizou-se o *framework* Hibernate[24].

Para representar o fluxo do sistema de diálogos projetado, utilizou-se um componente de código aberto chamado JGraphX[25].

Para o módulo de reconhecimento de voz existente no *wizard*, utilizou-se a JLaPSAPI (OLIVEIRA et al., 2011), citada na seção 2.1.4, para conectar o DialogBuilder com o *software* Coruja. Os detalhes deste módulo serão discutidos na seção 3.2.1. É válido ressaltar que o preenchimento de campos de tela não pode ser feito através de voz, sendo sempre feito manualmente pelo usuário: o reconhecedor do DialogBuilder é utilizado somente no *wizard*.

Para a síntese de voz existente no *wizard*, optou-se pelo uso de arquivos de áudio pré-gravados, uma vez que o domínio do *wizard* é controlado e o esforço para

---

[19] http://apstel.com. Acesso em: 24 nov. 2013.
[20] http://www.oracle.com/technetwork/java/javase/overview/index.html. Acesso em: 24 nov. 2013.
[21] http://docs.oracle.com/javase/tutorial/uiswing. Acesso em: 24 nov. 2013.
[22] http://netbeans.org. Acesso em: 24 nov. 2013.
[23] http://www.h2database.com. Acesso em: 24 nov. 2013.
[24] http://www.hibernate.org. Acesso em: 24 nov. 2013.
[25] http://www.jgraph.com/jgraph.html. Acesso em: 24 nov. 2013.

a gravação do áudio era aceitável, principalmente considerando que a síntese de voz não é o serviço principal oferecido pelo DialogBuilder.

O aplicativo foi desenvolvido no sistema operacional Ubuntu[26] e só pode ser executado em ambientes com sistema operacional do tipo Linux.

As versões de cada tecnologia utilizada no DialogBuilder são apresentadas na Tabela 3.1.

Tabela 3.1 – Tecnologias utilizadas no DialogBuilder e suas versões.

Tecnologia	Versão Utilizada
Java JDK	7u11
NetBeans	7.2.1
H2	1.3.170
Hibernate	3.6
JGraphX	1.10.4
Ubuntu	12.04 LTS
JLaPSAPI	1.7.2.2
Coruja	1.7.1

3.2.1 Módulo de Reconhecimento de Voz do *Wizard*

Para reconhecimento de voz durante o *wizard*, utilizou-se o *software* Coruja, através da JLaPSAPI. Vale dizer que o reconhecimento de voz é utilizado para manter o diálogo com o usuário e, mesmo durante o *wizard*, sempre que uma tela deve ser preenchida, isto deve ser feito manualmente.

Basicamente, ao se iniciar o *wizard*, um único reconhecedor é criado para ser utilizado durante todo o processo. O índice de confiança utilizado nesse reconhecedor é de 50%, o padrão provido pela JLaPSAPI. Além disso, não se utiliza modo ditado, existindo sempre uma gramática ativa para o reconhecimento.

[26] http://www.ubuntu.com. Acesso em: 24 nov. 2013.

Ao todo, foram criadas quatro gramáticas: uma para ações de configuração de módulos dentro do *wizard* (os detalhes do funcionamento do *wizard* são apresentados na seção 3.3.5.2); outra para perguntas que demandem respostas “sim” ou “não”; outra para perguntas sobre o tipo de pergunta desejada pelo usuário; e uma com as opções básicas de ações para seleção do usuário no *wizard*.

O funcionamento do reconhecedor é simples: cada ação no *wizard* tem associada a si uma gramática e, após dar ao usuário instruções ou realizar uma pergunta, o reconhecedor é iniciado e fica aguardando a resposta. Uma vez que uma resposta é aceita, o texto transcrito é repassado de volta à ação no *wizard* que pausa o reconhecedor, realiza as tarefas necessárias de acordo com a resposta do usuário e encaminha este para a próxima ação.

Vale dizer que nenhum artifício de confirmação explícita de respostas foi utilizado nesta primeira versão do DialogBuilder. Dessa forma, em caso de problema de reconhecimento, não há como o usuário corrigir sua solicitação ou retornar ao estado anterior. O desenvolvimento de mecanismos de tratamento de erro tanto no módulo de reconhecimento de voz quanto no *wizard* é um trabalho a ser realizado futuramente.

3.3 FUNCIONALIDADES

3.3.1 Tela Principal

Ao se iniciar o DialogBuilder, é apresentada ao usuário a tela principal do programa. Nela, o usuário tem acesso ao menu principal de funcionalidades e também visualiza a região de seleção dos módulos existentes e a região de visualização do fluxo do diálogo sendo gerado, ambas desabilitadas, pois nenhum projeto ainda foi aberto para edição. A Figura 3.1 mostra a tela principal logo que o DialogBuilder é iniciado.

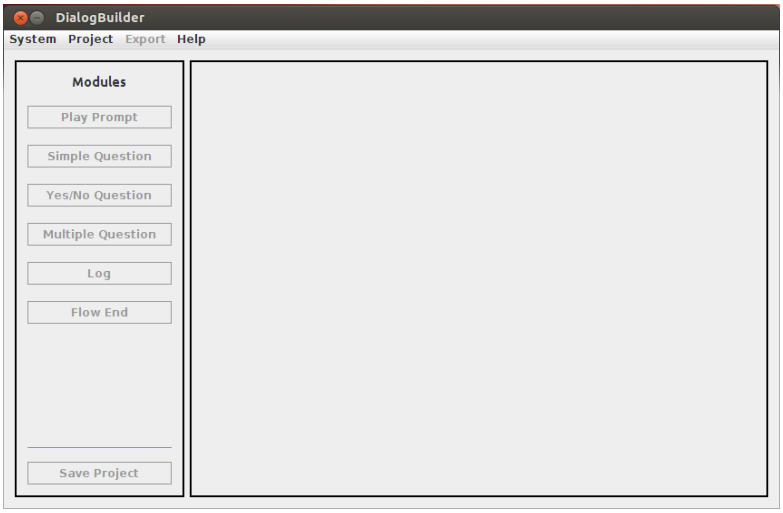


Figura 3.1 – Tela principal do DialogBuilder.

Existem quatro opções no menu principal: “System”, “Project”, “Export” e “Help”. A opção “System” traz três funcionalidades gerais do sistema: seleção da linguagem a ser utilizada (“Select Language”) pelo sistema; execução do *wizard* para projeto de sistema de diálogo (“Start Wizard”); e finalização da aplicação (“Exit”). Já a opção “Project” traz as funcionalidades relativas aos projetos de sistema de diálogo: criar novo projeto (“New”); abrir um projeto salvo anteriormente (“Open”); editar informações gerais de um projeto aberto (“Edit”); excluir um projeto aberto (“Delete”); e fechar um projeto aberto (“Close”). As funcionalidades “Edit”, “Delete” e “Close” só ficam habilitadas se um projeto de sistema de diálogo estiver aberto. A opção “Export” traz uma única funcionalidade que é a exportação do projeto de sistema de diálogo do DialogBuilder para código Asterisk (“Export to Asterisk”), possibilitando a implantação do sistema de diálogo projetado. Esta opção só fica habilitada quando há um projeto aberto. A última opção do menu principal é “Help”, que traz funcionalidades auxiliares ao sistema: exibição de informações gerais sobre o DialogBuilder (“About DialogBuilder”) e exibição de um tutorial básico sobre como fazer a implantação do sistema de diálogo projetado e exportado no Asterisk (“Deploy to Asterisk Tutorial”). A Figura 3.2 mostra os quatro menus expandidos.

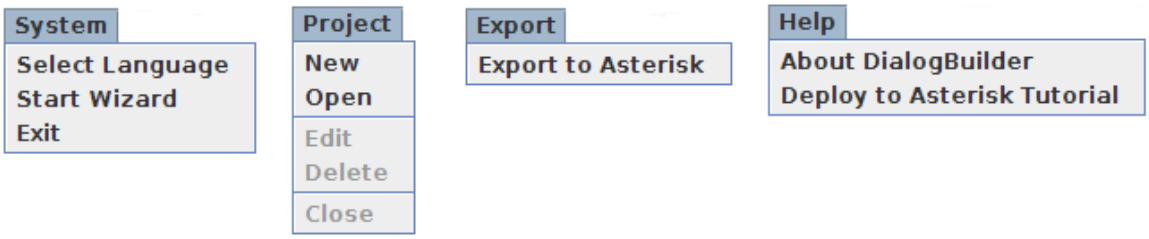


Figura 3.2 – Menus expandidos.

Cada opção do menu principal será detalhada nos próximos tópicos, buscando apresentar com detalhes o funcionamento do DialogBuilder.

3.3.2 Menu “Project”

No menu “Project”, inicialmente, o usuário pode criar um novo projeto de sistema de diálogo ou abrir um projeto já existente. Ao clicar para abrir um projeto já existente, a tela mostrada na Figura 3.3 é exibida contendo a lista de todos os projetos anteriormente salvos.

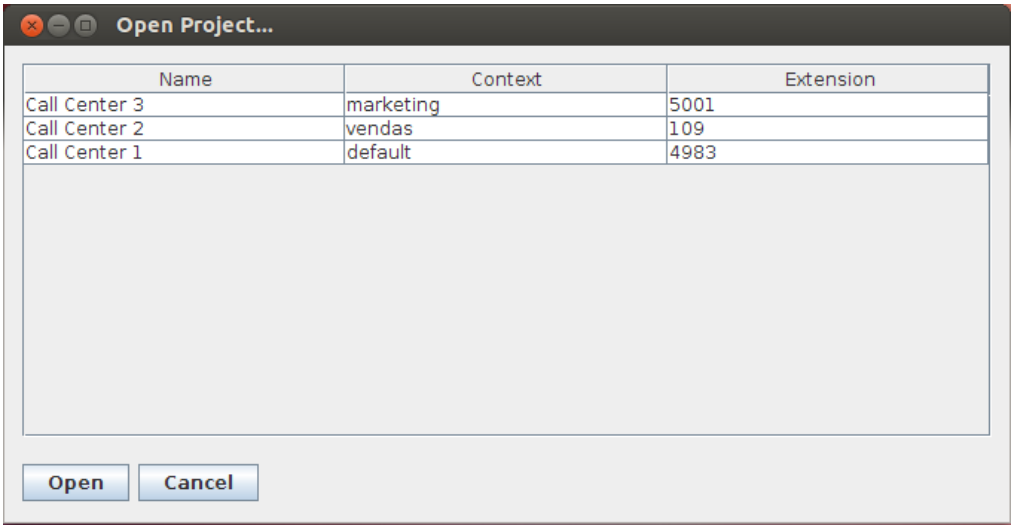


Figura 3.3 – Tela de consulta de projetos salvos anteriormente.

Ao se selecionar um projeto existente e clicar em “Open”, o projeto é aberto e carregado no sistema, conforme será explicado na seção 3.3.2.1.

Se o usuário clicar para criar um novo projeto, a tela da Figura 3.4 será apresentada a ele. Nesta tela, o usuário poderá dar um nome ao seu projeto e precisará definir o contexto do Asterisk para o qual o projeto será feito e a extensão que responderá pelo diálogo projetado. Aqui é importante frisar que no estágio atual do DialogBuilder, cada projeto de sistema de diálogo é feito para um único contexto e uma única extensão. Isso significa também que o código Asterisk gerado conterá somente um plano de discagem para esta extensão, neste contexto. Portanto, caso o usuário deseje utilizar mais de um contexto diferente em sua aplicação ou então mais de uma extensão, ele precisará projetar diversos sistemas de diálogos, um para cada par contexto/extensão, e juntar os diversos códigos gerados pelo DialogBuilder manualmente, conforme será explicado com mais detalhes na seção 3.3.4.2.

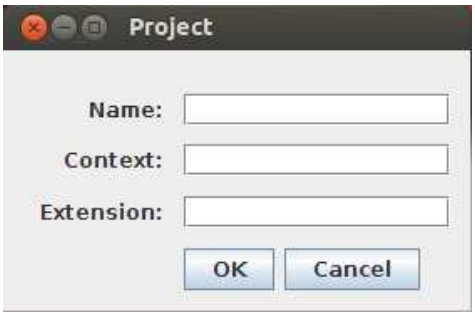


Figura 3.4 – Tela de informações gerais do projeto.

Quando o usuário clica em “OK” após o preenchimento das informações gerais, um novo projeto é gerado e carregado no sistema, conforme será explicado na seção 3.3.2.1.

Quando um projeto é criado ou aberto, as funcionalidades disponíveis no menu “Project” mudam e somente fica possível editar o projeto aberto, excluí-lo e fechá-lo. Ao se selecionar a opção para editar um projeto, a tela da Figura 3.4 é novamente exibida, porém com as informações do projeto já cadastradas anteriormente. Dessa forma, o usuário pode alterá-las conforme for conveniente. Ao se selecionar a opção para fechar um projeto aberto, o sistema irá limpar da memória em execução as informações do projeto, limpando a tela principal e voltando a permitir, através do menu “Project”, somente que um novo projeto seja criado ou um projeto existente seja aberto. Vale ressaltar que o que havia sido salvo do projeto anteriormente continua guardado na base de dados. Por último, ao se

selecionar a opção para excluir o projeto aberto, o sistema primeiramente fecha o projeto, conforme mencionado anteriormente e, em seguida, limpa o seu registro também da base de dados.

3.3.2.1 Carregando um Projeto

Um projeto é uma entidade que possui, além das informações gerais de nome, contexto e extensão, um fluxo de diálogo. Este fluxo é tratado pelo sistema em uma estrutura de grafo direcionado, onde cada vértice contém um módulo que representa uma ação que o sistema de diálogo deve executar durante a conversa com o cliente; e as arestas representam o fluxo propriamente dito entre as diversas ações que o sistema deve executar.

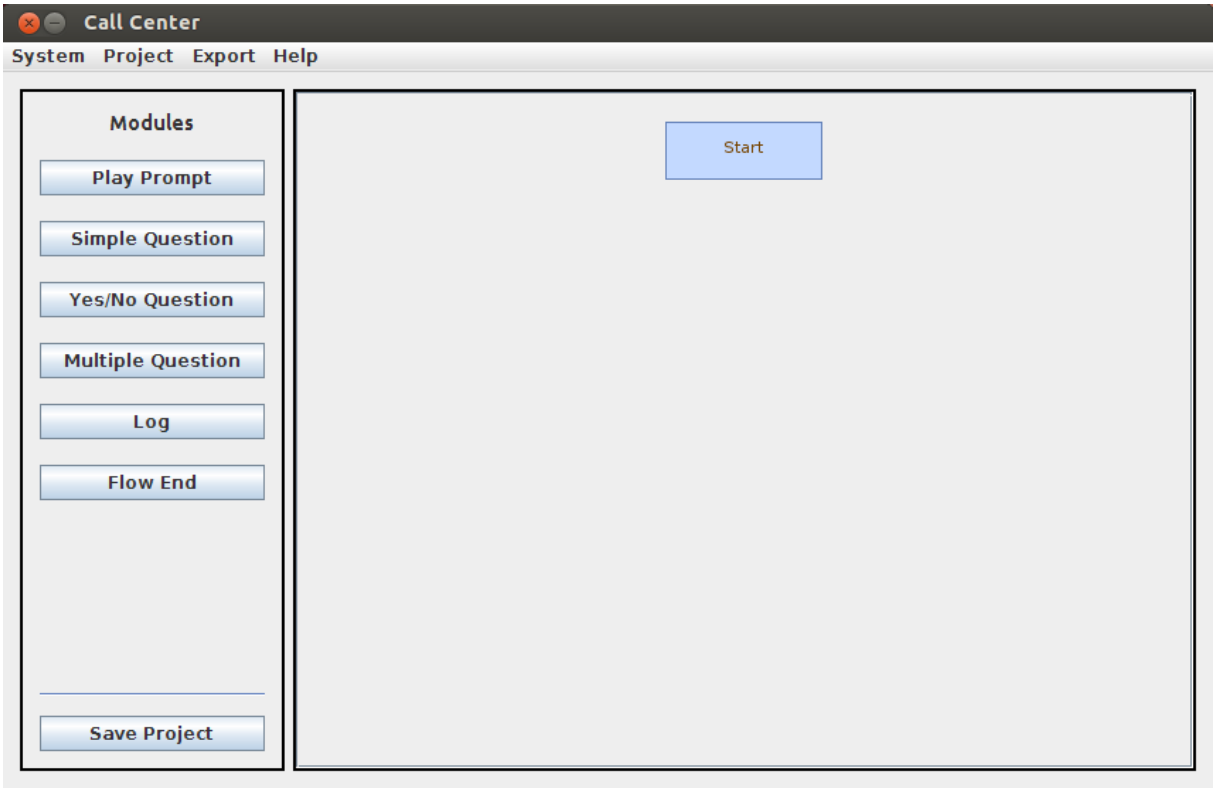
Todas as informações deste grafo, assim como as configurações individuais de cada módulo, são mantidas pelo DialogBuilder em base de dados com uma estrutura de dados adequada para tal, porém não inteligível facilmente para o usuário do sistema. Sendo assim, para que o usuário possa trabalhar em tal fluxo é necessário gerar uma visualização gráfica editável desse. Para isso, a biblioteca JGraphX foi utilizada. Portanto, o DialogBuilder faz o mapeamento das informações do fluxo na estrutura de dados utilizada para armazenamento com o formato necessário para visualização na biblioteca gráfica. Dessa forma, alterações oriundas no modelo de dados no DialogBuilder irão refletir na visualização gráfica, assim como o contrário também: sempre que o usuário atualizar o fluxo através de sua visualização gráfica, o modelo de dados também será modificado.

A partir do já exposto, o processo de carregar um projeto pode ser descrito em passos:

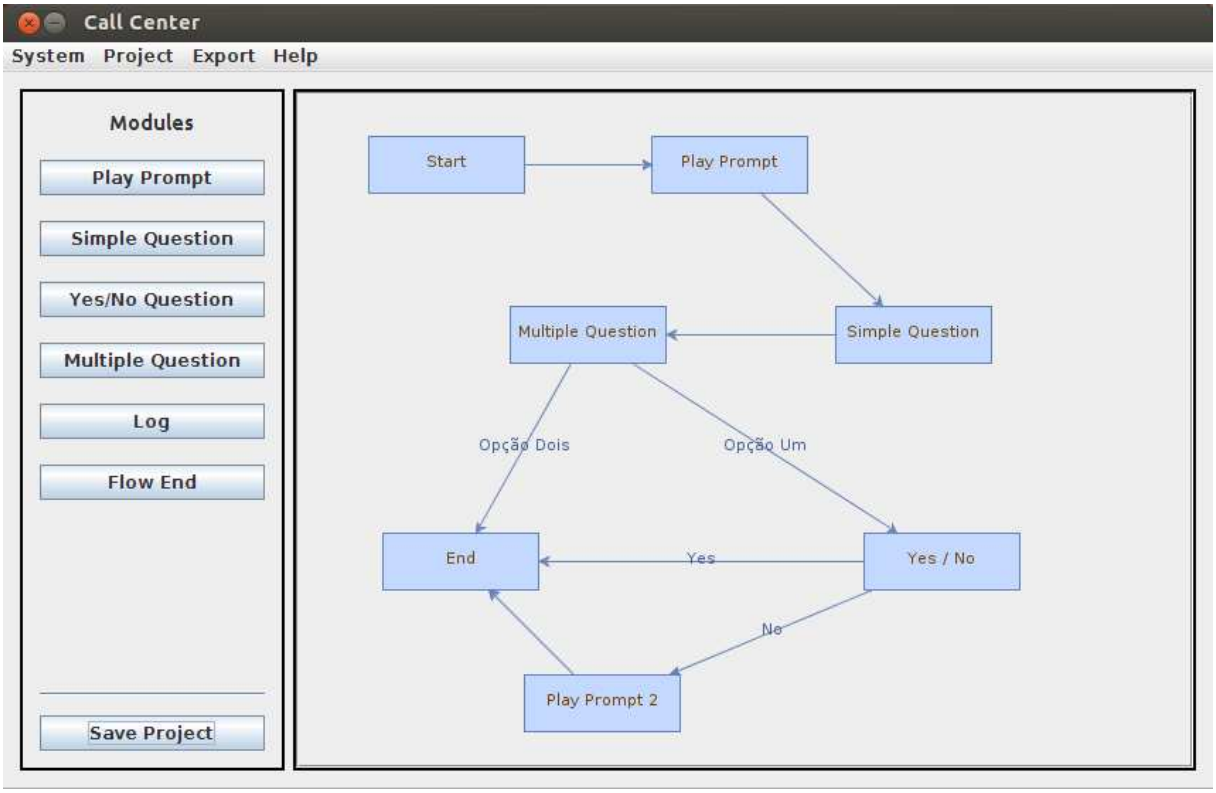
1) Colocar em memória os dados do projeto a ser carregado;
2) Atualizar o título da tela principal do sistema com o nome do projeto;
3) Atualizar os menus, desabilitando aqueles que não são permitidos com um projeto aberto e habilitando os demais;

4) Habilitar a região de escolha de módulos para que o usuário possa trabalhar no fluxo relativo ao projeto sendo carregado;
5) Realizar o mapeamento das informações de fluxo de diálogo do projeto sendo carregado para o modelo gráfico do JGraphX, configurando-o para possibilitar a edição do usuário.

Vale ressaltar que mesmo no caso de um novo projeto, todos os passos acima serão executados. A diferença é que o fluxo conterá somente um único vértice representando seu início, isto é, o recebimento da ligação pelo sistema de diálogo. A Figura 3.5 mostra a tela principal do DialogBuilder em duas situações: logo após um novo projeto ser criado e logo após um projeto já existente ser aberto.



(a)



(b)

Figura 3.5 – Tela principal do DialogBuilder quando: (a) um novo projeto de sistema de diálogo é criado; e (b) um projeto de sistema de diálogo já existente é aberto.

3.3.2.2 Montando o Fluxo de Diálogo

A partir do vértice contendo o módulo de início do diálogo, o usuário deve montar o fluxo da conversa agregando novos módulos (vértices) a este e conectando-os da forma correta para obter o resultado desejado.

Para incluir um novo módulo no fluxo, o usuário deve clicar no botão correspondente ao módulo desejado constante do menu de módulos. Um novo vértice será então exibido na visualização gráfica do fluxo de diálogo e a tela de configuração do módulo será exibida. Os módulos disponíveis atualmente no DialogBuilder serão detalhados na seção 3.3.3.

Uma vez inserido o vértice novo no fluxo, o usuário pode posicioná-lo da melhor forma clicando e arrastando o vértice, como é mostrado na Figura 3.6. Para

editar a configuração do módulo, o usuário deve dar um clique simples sobre o vértice representativo do módulo. E, por último, para conectar dois vértices, o usuário deve clicar no vértice do qual deve partir a conexão (quando sua borda estiver em destaque) e arrastar a aresta até o vértice que deve suceder o anterior, conforme pode ser visto na Figura 3.7. Para alguns módulos específicos, o ato de criar uma aresta pode implicar no aparecimento de uma tela de configuração da aresta, conforme será detalhado na seção 3.3.3.

Após montar todo o fluxo, o usuário deve salvar as alterações realizadas neste clicando no botão “Save Project”.

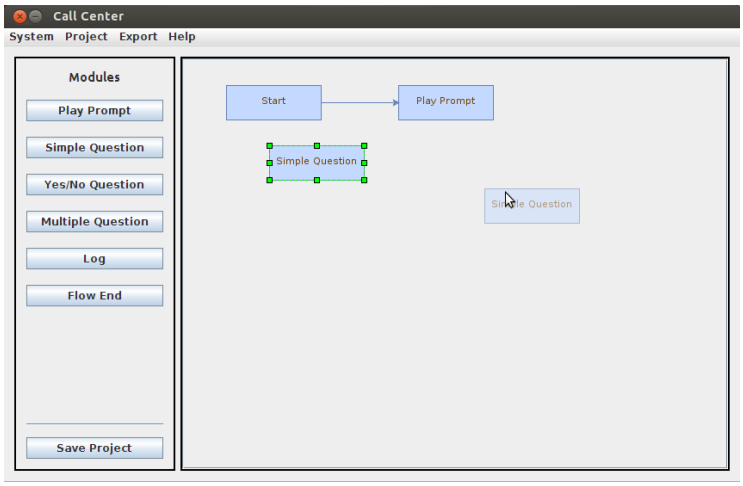


Figura 3.6 – Movendo um vértice no fluxo.

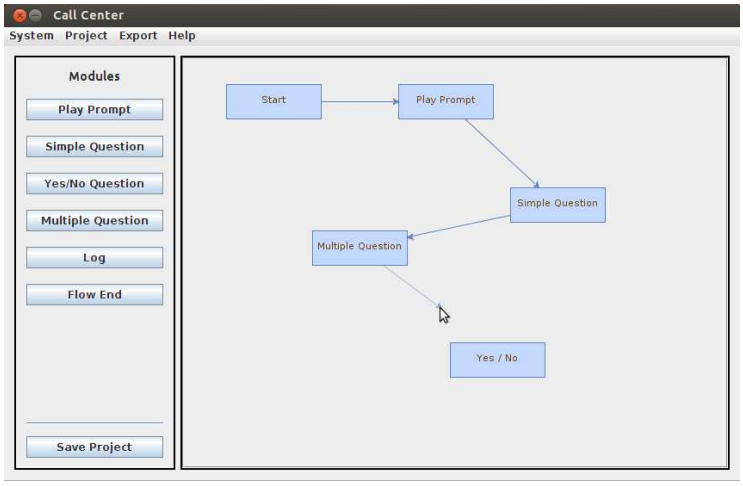


Figura 3.7 – Criando uma aresta no fluxo.

3.3.3 Módulos

Nesta seção, serão apresentados os módulos atualmente disponíveis no DialogBuilder. Eles serão explicados e suas telas de configuração serão detalhadas.

3.3.3.1 Play Prompt

O módulo “Play Prompt” representa a ação de executar um *prompt* em meio ao diálogo. Um *prompt* nada mais é que um arquivo de áudio. Portanto, este módulo pode ser utilizado para que o sistema de diálogo toque uma música de espera, diga uma mensagem de boas vindas ou informe ao usuário algum aviso.

Ao ser incluído no diálogo, este módulo precisa ser configurado através da tela mostrada na Figura 3.8. Nesta tela, o usuário deve informar o nome que deseja dar para esta instância do módulo “Play Prompt”, como, por exemplo, “Agradecimento”; e também selecionar que arquivo de áudio em seu computador deve ser executado.

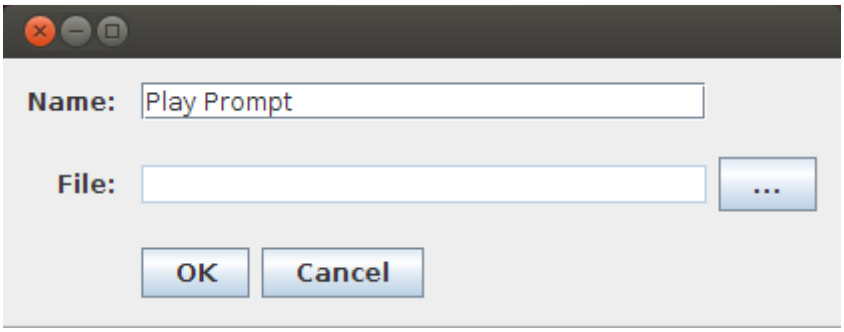


Figura 3.8 – Tela de configuração do módulo “Play Prompt”.

3.3.3.2 Simple Question

O módulo “Simple Question” representa a ação de realizar uma pergunta simples ou aberta em meio ao diálogo. Uma pergunta simples é uma pergunta sem opções fixas de respostas, o que não significa que não haja um conjunto possível de respostas a serem esperadas pelo reconhecedor. Por exemplo, “qual o nome da cidade em que você vive?” é uma pergunta simples, pois não existem opções dadas ao usuário como resposta. Porém, há um conjunto restrito de respostas possíveis que é o nome de todas as cidades do Brasil, por exemplo, ou mesmo um conjunto menor de cidades caso o sistema de diálogo seja feito somente para usuários de uma determinada região.

A tela de configuração deste módulo está exibida na Figura 3.9. Nesta tela, o usuário deve informar um nome para esta instância do módulo; informar um arquivo de áudio contendo a pergunta a ser feita; e também informar uma lista de possíveis respostas, separadas por ponto-e-vírgula.

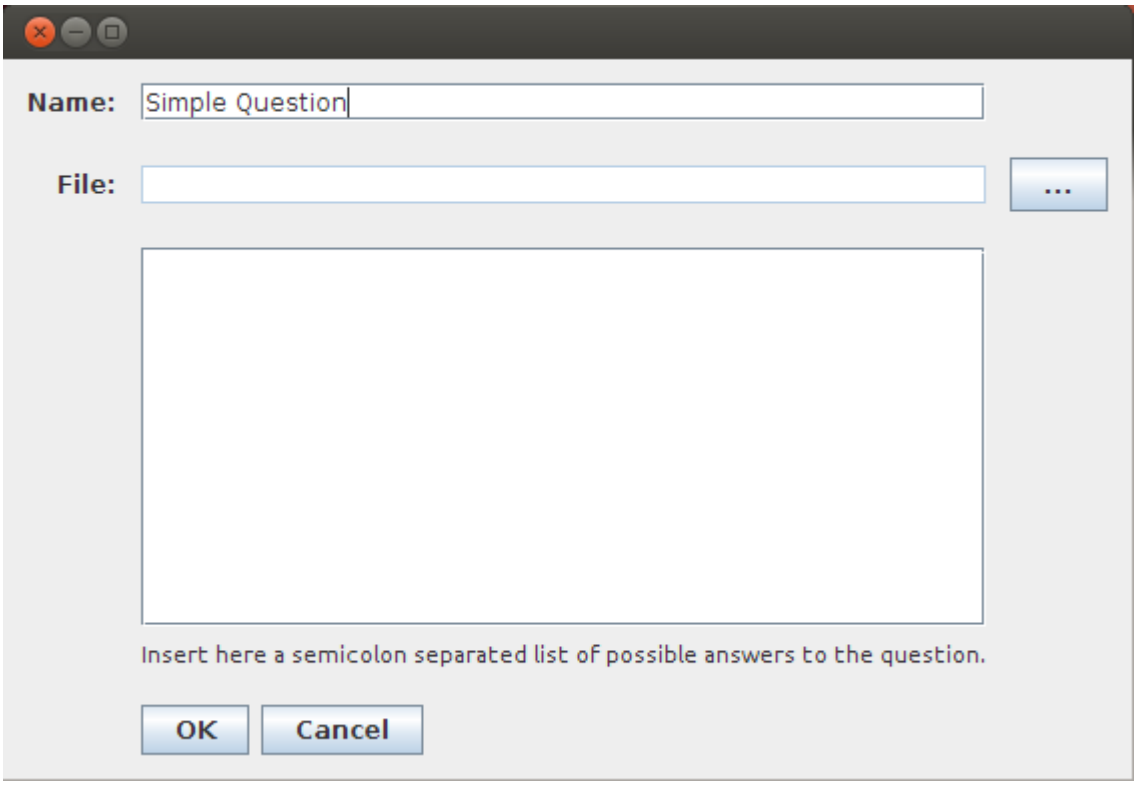


Figura 3.9 – Tela de configuração do módulo “Simple Question”.

3.3.3.3 Multiple Question

O módulo “Multiple Question” representa uma ação de realizar uma pergunta com opções definidas de respostas. Um exemplo desse tipo de pergunta é: “Qual o produto que você deseja adquirir? Celular, computador ou *tablet*?”, onde as opções de respostas seriam “celular”, “computador” e “*tablet*”.

A tela de configuração deste módulo está exibida na Figura 3.10. Nesta tela, o usuário deve informar um nome para esta instância do módulo; informar um arquivo de áudio contendo a pergunta a ser feita; informar a quantidade de opções de respostas que deseja (variando de 2 a 9) e também informar quais serão estas opções nas caixas de texto correspondentes.

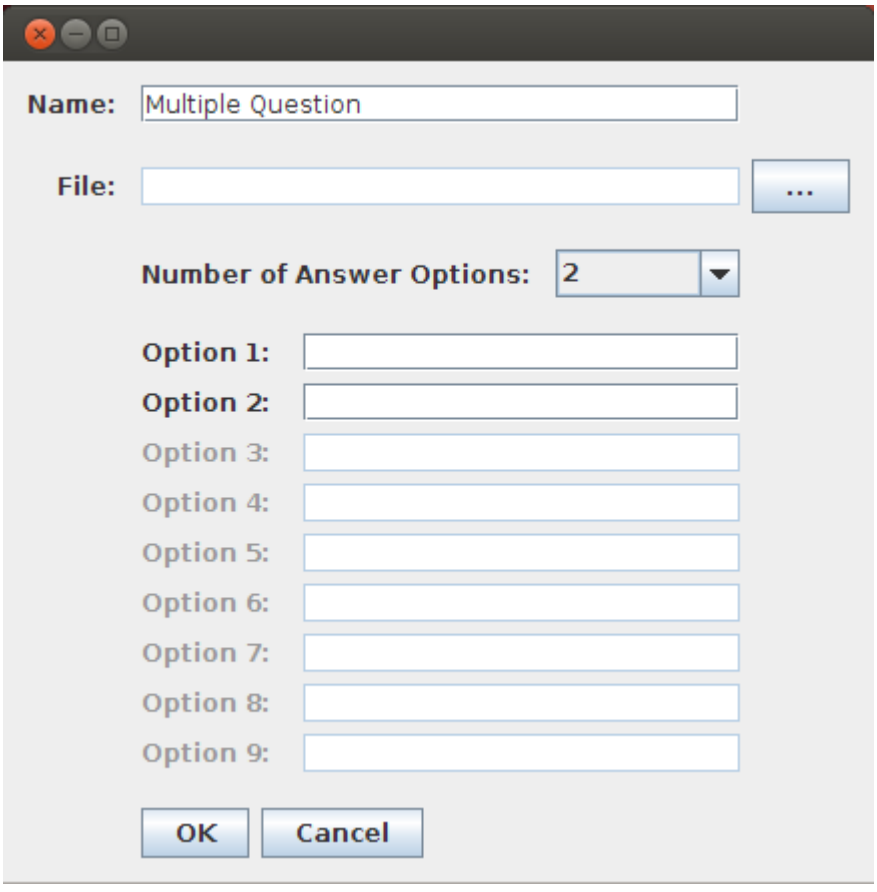


Figura 3.10 – Tela de configuração do módulo “Multiple Question”.

Neste tipo de pergunta, cada opção de resposta pode gerar um encaminhamento diferente no fluxo do diálogo, isto é, de cada vértice contendo um

módulo “Multiple Question” sairão tantas arestas quanto forem as opções de respostas possíveis. No exemplo dado anteriormente, existiriam três arestas: uma representando a resposta “celular”, uma representando a resposta “computador” e outra representando a resposta “*tablet*”. Para que o usuário possa identificar tais arestas, sempre que ele conectar um vértice contendo o módulo “Multiple Question” a outro vértice, uma tela como a da Figura 3.11 será exibida, contendo uma lista de opções de respostas configuradas para o módulo e que ainda não foram atribuídas a alguma aresta anteriormente. O usuário então poderá selecionar que a resposta que deve ser associada à aresta que ele está criando.

Figura 3.11 – Tela para associação de aresta com opção de resposta correspondente.

3.3.3.4 Yes/No Question

O módulo “Yes/No Question” representa uma ação de realizar uma pergunta cuja resposta é “sim” ou “não”. Este módulo é uma especialização do módulo “Multiple Question”, explicado na seção 3.3.3.3, pois representa uma pergunta com duas opções de resposta. Ele foi isolado como um módulo à parte porque ocorre com bastante frequência e, dessa forma, ajuda o projetista do diálogo em sua tarefa.

A tela de configuração deste módulo está exibida na Figura 3.12. Nesta tela, o usuário deve informar um nome para esta instância do módulo e um arquivo de áudio contendo a pergunta a ser feita. Isso é suficiente porque as opções de respostas já são conhecidas.

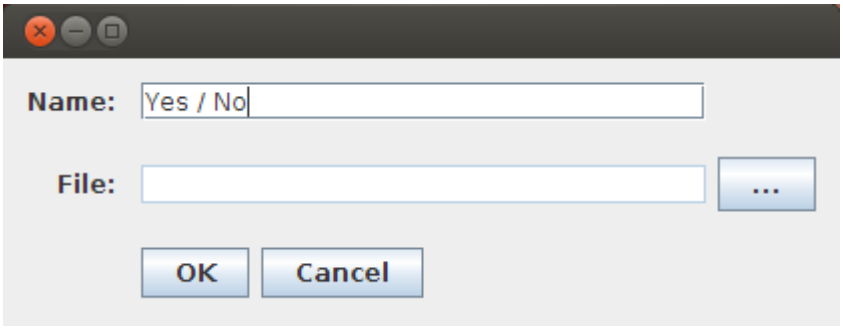


Figura 3.12 – Tela de configuração do módulo “Yes/No Question”.

Tal qual no módulo “Multiple Question”, quando o projetista conectar um vértice contendo este módulo a outro, ele precisará selecionar a que opção de resposta essa aresta corresponde.

3.3.3.5 Log

O módulo “Log” representa a ação de registrar no arquivo de *log* padrão do Asterisk a resposta do usuário a uma pergunta simples. Ele não necessita de configuração é deve ser utilizado somente conectado a um vértice contendo o módulo “Simple Question”.

A ideia deste módulo é ser uma semente de futuros módulos a serem ainda criados que possibilitem que a resposta do usuário a perguntas simples sejam tratadas das mais diversas formas. Por exemplo, se o sistema de diálogo perguntar “Qual o nome da cidade que você vive?” para o cliente, a resposta poderia ser usada para acessar uma base de dados e obter uma informação específica a ser informada para aquele cliente, relativa à cidade em que ele vive.

3.3.3.6 Flow End

O módulo “Flow End” representa a ação de encerrar a chamada. Ele não necessita de configuração e pode ser conectado a qualquer vértice.

3.3.4 Menu “Export”

A única funcionalidade da opção “Export” do menu principal é exportar o projeto aberto, gerando todos os insumos necessários para utilizá-lo no Asterisk (“Export to Asterisk”). Vale ressaltar que o DialogBuilder não configura diretamente o Asterisk, mas apenas gera tudo aquilo que é necessário para tal. Atualmente, o usuário deve manualmente mover o material gerado (gramáticas, arquivos de mídia e *dialplan*) para as pastas correspondentes do Asterisk. Esse processo manual será explicado na seção 3.3.4.2.

Os insumos para o Asterisk relativos ao projeto de sistema de diálogo sendo exportado são gerados dentro da pasta “*target*”, em um diretório com o nome do projeto (sem os espaços em branco). Sendo assim, o primeiro passo para exportação é limpar tal diretório, caso ele exista, ou criá-lo, caso contrário.

Em seguida, o arquivo “extensions.conf”, que contém do *dialplan*, é criado no diretório alvo da exportação e o contexto (informado na criação do projeto) é inserido, entre colchetes, na primeira linha deste arquivo, conforme sintaxe do *dialplan*.

Após isso, o grafo representativo do fluxo de diálogo é percorrido em profundidade e o *dialplan* é atualizado em cada vértice, de acordo com o módulo contido neste. A forma como cada módulo atualiza o *dialplan* é discutida na próxima seção.

3.3.4.1 Regras de Exportação dos Módulos

O primeiro módulo que é exportado é o módulo de início de diálogo (“Flow Start”). Este módulo inclui no *dialplan* a seguinte linha de código:

```
exten => extension, 1, answer()
```

A palavra *extension* representa, aqui e no restante desta seção, a extensão informada na criação do projeto. O número “1” representa a prioridade desta ação. Para o módulo inicial, ela será sempre “1” porque o módulo inicial é sempre único por diálogo e sempre o primeiro módulo a ser executado. Porém, a cada nova ação exportada, esta prioridade é incrementada em uma unidade. Sendo assim, daqui por diante a prioridade em cada ação será representada pela palavra *priority*.

Na exportação do módulo “Play Prompt”, a primeira ação executada é a cópia do arquivo de áudio selecionado para execução do local onde ele foi selecionado para a pasta *media* dentro do diretório para onde o projeto de sistema de diálogo está sendo exportado. Em seguida, o seguinte código é inserido no *dialplan*:

```
exten => extension, priority, playback(nome_do_arquivo)
```

No código acima, o parâmetro “nome_do_arquivo” deve ser substituído pelo nome do arquivo de áudio a ser executado sem sua extensão.

Na exportação do módulo “Simple Question”, a primeira ação executada é a geração da gramática relativa à pergunta. Para gerar esta gramática, a lista de respostas possíveis configurada no módulo é transformada em uma gramática do tipo SAPI XML da seguinte forma:

```
<GRAMMAR>
<RULE TOPLEVEL="ACTIVE" ID="1">
      <PHRASE>primeira_opcao_resposta</PHRASE>
      <RETURN> primeira_opcao_resposta </RETURN>
</RULE>
...
</GRAMMAR>
```

Esta gramática, que contém uma regra para cada opção de resposta configurada no módulo, é então compilada através do *script pysapixml2julius*, executado pelo próprio DialogBuilder. Os arquivos resultantes da compilação da gramática são então movidos para a pasta *grammar* dentro do diretório para onde o sistema de diálogo está sendo exportado.

Após a geração da gramática, o arquivo de áudio é copiado para a pasta *media* do diretório de exportação, da mesma forma como é realizado para o módulo “Play Prompt”.

Por último, o código inserido no *dialplan* para o módulo “Simple Question” é:

```
exten => extension, priority, playback(nome_do_arquivo)
exten => extension, priority, EAGI(adintool.py,
         julius.jconf, nome_da_gramática, 6, 1, ,
         /tmp/audio.wav)
exten => extension, priority, Set(ANSWER=${RECRET:0})
```

Vale notar que, para este módulo, o *dialplan* toca o áudio da pergunta; em seguida, ele inicia o reconhecedor coruja utilizando a gramática gerada (o “nome_da_gramática” é o mesmo nome do módulo, sem espaços em branco); e, por último, ele grava a resposta da pergunta, obtida através da variável RECRET, que é onde o reconhecedor grava a informação contida na *tag* RETURN da gramática, na variável ANSWER de forma que esta informação possa ser utilizada posteriormente por outros módulos como o módulo de “Log”.

Na exportação do módulo “Log”, tudo o que o sistema faz é inserir no *dialplan* o seguinte código:

```
exten => extension, priority, log(NOTICE,${ANSWER})
```

Ou seja, a ação do módulo “Log” é simplesmente registrar em *log*, com nível NOTICE, a resposta do cliente a uma pergunta simples. Este módulo tem o propósito de ilustrar o uso da resposta a uma pergunta simples para outras finalidades.

Na exportação do módulo “Multiple Question”, de forma semelhante ao módulo de perguntas simples, o primeiro passo é gerar uma gramática com as opções de respostas configuradas no módulo e copiar os arquivos resultantes para a

pasta *grammar* dentro do diretório de exportação do sistema de diálogo. Em seguida, o arquivo de áudio da pergunta é copiado para a pasta *media* do mesmo diretório.

O código relativo ao módulo é então inserido no *dialplan* (aqui se supõe que existem *n* opções de resposta possíveis configuradas no módulo):

```
exten => extension, priority, playback(nome_do_arquivo)
exten => extension, priority, EAGI(adintool.py,
         julius.jconf, nome_da_gramática, 6, 1, ,
         /tmp/audio.wav)
exten => extension, priority, Set(CONF=${RECRET:0})
exten => extension, priority, GotoIf($[${CONF} =
         opção_de_resposta_1]?PRIORITYOPTION1:priority+1
exten => extension, priority+1, GotoIf($[${CONF} =
         opção_de_resposta_2]?PRIORITYOPTION2:priority+2
exten => extension, priority+2, GotoIf($[${CONF} =
         opção_de_resposta_3]?PRIORITYOPTION3:priority+3
...
exten => extension, priority+n-1, GotoIf($[${CONF} =
         opção_de_resposta_n]?PRIORITYOPTIONn:priority+n
```

De acordo com o código, o sistema toca o arquivo de áudio, inicia o reconhecedor com a gramática gerada, guarda a opção de resposta falada pelo cliente e então faz uma série de comparações com o objetivo de enviar o fluxo de diálogo para a prioridade correspondente à opção escolhida. Este código inicialmente gerado na exportação ainda deve ser atualizado, na medida em que a exportação vai avançando, para que cada PRIORITYOPTIONX seja substituído pelo número de prioridade onde o ramo do grafo de diálogo relativo à opção de resposta X começa a ser exportado. Como não se pode prever a quantidade de ações que o sistema deverá executar para cada opção de resposta, não é possível saber, de antemão, tais valores de prioridades, exigindo essa atualização posterior.

Na exportação do módulo “Yes/No Question” o processo é semelhante ao módulo de perguntas com múltiplas respostas. A diferença principal reside no fato de que a gramática utilizada é a mesma apresentada na seção 2.3.4.1. Ela somente é compilada no momento da exportação e os arquivos resultantes são colocados na pasta *grammar* do diretório para o qual o sistema de diálogo está sendo exportado. O arquivo de áudio é copiado para a pasta *media* deste mesmo diretório. E o código inserido no *dialplan* do diálogo é:

```
exten => extension, priority, playback(nome_do_arquivo)
exten => extension, priority, EAGI(adintool.py,
         julius.jconf, yesNoGrammar, 6, 1, ,
         /tmp/audio.wav)
exten => extension, priority, Set(CONF=${RECRET:12:1})
exten => extension, priority, GotoIf($[${CONF} = 1]?
         YESPRIORITY:NOPRIORITY
```

Mais uma vez, o código gerado deverá ser atualizado na medida em que a exportação avançar para que YESPRIORITY e NOPRIORITY sejam substituídos pela prioridade em que se iniciam os ramos do fluxo relativos às opções sim (“Yes”) e não (“No”), respectivamente.

Por último, na exportação do módulo “Flow End”, toda ação necessária é incluir o seguinte código no *dialplan* do sistema de diálogo, para que este encerre a chamada:

```
exten => extension, priority, hangup()
```

3.3.4.2 Implantando o Sistema de Diálogo no Asterisk

Uma vez que a exportação tenha sido concluída, todos os artefatos necessários para a implantação do sistema de diálogo no Asterisk estarão disponíveis em um diretório homônimo ao projeto dentro da pasta *target* da aplicação.

O processo para implantar o sistema no Asterisk está descrito a seguir:

1) Copie os arquivos de mídia, disponíveis na pasta *media* do diretório para onde o sistema de diálogo foi exportado, para a pasta de sons do Asterisk. Numa instalação padrão, esta pasta é **`/var/lib/asterisk/sounds`**;
2) Copie os arquivos de gramática, disponíveis na pasta *grammar* do diretório para onde o sistema de diálogo foi exportado, para a pasta de gramáticas utilizadas pelo Coruja no Asterisk. Por padrão, esta pasta é **`/usr/src/coruja-asterisk/grammars`**;

3) Copie o arquivo *extensions.conf*, disponível na raiz do diretório para onde o sistema de diálogo foi exportado, para a pasta do Asterisk em que os arquivos de configuração ficam. Numa instalação padrão, esta pasta é **`/etc/asterisk`**;
4) Entre no console do Asterisk para recarregar os *scripts dialplan*: para isso, digite "**`asterisk –vvvr`**" no terminal e, em seguida, execute o comando "**`dialplan reload`**" no console do Asterisk.

Após estes passos, o sistema de diálogo projetado já estará sendo executado no Asterisk.

Vale ressaltar que caso o usuário deseje utilizar mais de uma extensão ou contexto em seu sistema de diálogo, hoje ele precisará criar dois projetos no DialogBuilder (um para cada par contexto/extensão) e mesclar os arquivos *extensions.conf* gerados, antes de proceder como no passo 3 acima. Além disso, todos os arquivos de mídia e todas as gramáticas de ambos os projetos deveriam ser copiados para as respectivas pastas nos passos 1 e 2.

3.3.5 Menu "System"

Há três funcionalidades disponíveis na opção "System" do menu principal. A mais óbvia delas é sair do sistema ("Exit"). As outras duas tratam da seleção da linguagem a ser utilizada no DialogBuilder ("Select Language"), sendo intimamente ligada à internacionalização do aplicativo; e do *wizard* para criação de projeto de sistema de diálogo ("Start Wizard").

3.3.5.1 Selecionando uma Linguagem

A tela de seleção de linguagem a ser utilizada no DialogBuilder está mostrada na Figura 3.13. Nela, o usuário deve escolher, dentre as opções de linguagens disponíveis para o sistema, qual deseja utilizar.

Figura 3.13 – Tela para seleção de linguagem a ser usada no DialogBuilder.

As linguagens disponíveis devem ser incluídas em um arquivo de propriedades chamado *languages.properties*, que fica localizado na raiz da pasta *resources* do sistema. Este arquivo deve conter uma lista de pares chave-valor, no qual a chave é o nome da linguagem (a ser exibida na tela da Figura 3.13) e o valor é o código desta.

Uma vez selecionada a linguagem, a ideia é que o DialogBuilder possa ser inteiramente internacionalizado para ela, o que ainda não está feito totalmente. O que já está implementado está descrito abaixo:

1) Para o módulo “Yes/No Question”, a gramática utilizada (apresentada na seção 2.3.4.1) é buscada pelo sistema com o nome “YesNoGrammar.xml” dentro de um diretório nomeado segundo o código da linguagem selecionada, que deve ficar dentro da pasta *resources* do DialogBuilder. Dessa forma, para que se possa utilizar este módulo em outra linguagem, basta que a gramática seja traduzida e um arquivo homônimo seja criado no diretório nomeado com o código da nova linguagem, também dentro da pasta *resources*;
2) Da mesma forma, os arquivos de áudio utilizados no *wizard* (que será descrito na seção 3.3.5.2) são sempre buscados no diretório nomeado

segundo o código da linguagem selecionada, possibilitando que o *wizard* possa ser traduzido para outras línguas.

Vale ressaltar que, caso o usuário não selecione uma nova linguagem, o DialogBuilder utiliza, por padrão, o Português Brasileiro, que é a única língua para o qual o sistema foi preparado até o momento.

Para uma internacionalização completa, os seguintes passos ainda seriam necessários:

1) Arrumar o mecanismo para que os *labels* da interface gráfica da aplicação também fossem traduzidos ao se selecionar uma nova língua (hoje, mesmo com a única língua disponível sendo o Português Brasileiro, os *labels* estão todos em Inglês);
2) Possibilitar que se pudesse configurar o decodificador Julius do Coruja (a ser utilizado no Asterisk) para outra língua e passar o novo arquivo de configuração do Julius dentro das pastas de recurso internacionalizadas do DialogBuilder (atualmente, sempre é usado o mesmo arquivo de configuração que está pronto para o Português Brasileiro);
3) Possibilitar que se pudesse configurar o reconhecedor de voz do *wizard*, a ser detalhado na seção 3.3.5.2, para também usar uma gramática de acordo com a língua selecionada (atualmente, a gramática utilizada é sempre a mesma e feita para o Português Brasileiro).

É importante frisar que o trabalho de internacionalizar o sistema não é do usuário final. O usuário final deve receber o sistema já pronto e, simplesmente, selecionar a língua que deseja utilizar. As configurações necessárias para que se possa usar o DialogBuilder em outra língua devem ser feitas por um desenvolvedor colaborador com conhecimento suficiente para traduzir os artefatos para este novo idioma e, daí sim, disponibilizar para o usuário final.

3.3.5.2 *Wizard*

O *wizard* em forma de diálogo oferecido pelo DialogBuilder tem a proposta de conduzir o usuário no processo de criação de seu projeto de sistema de diálogo. Tal condução é feita através de um diálogo falado entre o usuário e o DialogBuilder.

Este *wizard* é um diferencial do DialogBuilder perante soluções semelhantes, tornando possível que um usuário crie seu próprio sistema de diálogo, sem necessidade de conhecimentos técnicos prévios. Vale ressaltar que a escolha de utilizar um diálogo com reconhecimento de fala em vez de interfaces mais tradicionais para *wizards* é uma aposta: mesmo que os demais tipos de interfaces possam apresentar maior eficiência hoje em dia, as interfaces de voz vêm evoluindo muito rapidamente, sendo tópico de diversas pesquisas de vanguarda, como discutido na seção 2.2, e largamente utilizadas em tecnologias de ponta como o Google Glass[27]. Dessa forma, acredita-se que a interação por voz possa trazer cada vez mais benefícios para este tipo de aplicação, até mesmo por se assemelhar com a forma de comunicação dos seres humanos.

Por último, é importante não confundir o uso do reconhecimento de voz dentro do Asterisk durante a execução do sistema de diálogo, com o uso do reconhecimento de voz no *wizard*.

3.3.5.2.1 Funcionamento do *Wizard*

O *wizard* deve ser dinâmico, pois cada projeto é diferente e não é possível prever de antemão todo o fluxo que o diálogo entre o usuário projetista e o DialogBuilder irá seguir. Além disso, ele também deve ser escalável, permitindo que outros módulos sejam inseridos futuramente no sistema. Com tais premissas, optou-se por utilizar um padrão de projetos de *software* chamado *Chain of Responsibility* e

[27] http://www.google.com/glass/start. Acesso em: 29 nov. 2013.

apresentado por Gamma et al. (1994). A principal característica deste padrão de projeto é que cada objeto do sistema possui a informação de qual o próximo objeto que deve ser executado.

Neste caso, os objetos são as ações que o DialogBuilder deve executar durante o *wizard*. Entende-se por ação o seguinte conjunto de ocorrências:

1) O sistema apresenta ao usuário uma tarefa ou realiza uma pergunta, através da execução de um arquivo de áudio, gravado anteriormente, específico desta ação;
2) Após apresentar a tarefa ou pergunta, o sistema inicia o reconhecedor de voz do *wizard* e aguarda um retorno do usuário: a confirmação de que a tarefa foi concluída ou a resposta da pergunta;
3) O usuário realiza a tarefa proposta e confirma sua execução ao sistema ou então, o usuário resposta à pergunta feita pelo sistema;
4) O sistema reconhece a fala do usuário e identifica, através do que foi dito, qual a próxima ação que deve ser tomada para continuar de maneira correta o *wizard*;
5) O sistema inicia a execução da próxima ação ou então, no caso de não haver próxima ação, encerra o *wizard*.

Como visto acima, cada ação (objeto do sistema) sabe qual deve ser a próxima ação executada de acordo com o retorno que recebe do usuário, caracterizando o padrão *Chain of Responsibility*, e, dessa forma, a cadeia de ações acontece dinamicamente, como desejado. Além disso, novas ações podem ser facilmente criadas e “encaixadas” no fluxo, bastando que se configurem corretamente as próximas ações necessárias a partir desta nova ação e também se modifique as ações que devem anteceder esta nova ação para que a execute, quando for o caso.

A seguir, cada ação criada será detalhada, especificando a tarefa ou pergunta (transcrição do áudio) associada e também as possíveis opções de próximas ações existentes.

a) Ação de Configuração do Projeto

Transcrição do áudio: **“Antes de iniciar o *wizard*, preencha o formulário com as informações gerais do diálogo. Ao terminar, clique em OK e diga ‘prossiga’.”**.

Assim que esta ação é executada, o áudio transcrito é executado e a tela de informações gerais do projeto (Figura 3.4) é exibida ao usuário. Este deve preenchê-la manualmente e confirmar, por voz, dizendo “prossiga”, que executou a tarefa.

Após a execução da tarefa associada a esta ação, o projeto será criado e o vértice inicial inserido na região de exibição do grafo do diálogo, que ficará como na Figura 3.5 (a).

Existe apenas uma próxima ação possível que é executar a primeira pergunta do *wizard*.

b) Ação da Primeira Pergunta

Transcrição do áudio: **“Olá! O que você gostaria que o sistema fizesse ao iniciar um atendimento para seu cliente? Diga ‘mensagem’ para o sistema executar uma mensagem de boas-vindas; ou diga ‘pergunta’ para o sistema realizar uma pergunta.”**.

Esta é a primeira pergunta do *wizard* e serve para que o DialogBuilder saúde o usuário projetista e o questione sobre a primeira ação que seu sistema de diálogo deve fazer ao iniciar o atendimento do cliente. Vale observar que esta ação não inclui a possibilidade de encerrar o atendimento, considerando que não faz sentido encerrar a ligação logo ao atendê-la.

Existem duas próximas ações possíveis. Caso o usuário diga “mensagem”, isso significa que um módulo “Play Prompt” deve ser inserido no fluxo e, portanto, a próxima ação do *wizard* deve ser de configuração de tal módulo. Caso o usuário diga “pergunta”, o sistema necessita saber se o usuário se refere a uma pergunta simples ou com múltiplas respostas, portanto a próxima ação é realizar um questionamento sobre o tipo de pergunta que o usuário deseja.

c) Ação de Configuração do Módulo “Play Prompt”

Transcrição do áudio: **“Preencha o formulário com o nome da mensagem e selecione o arquivo que contém a mensagem a ser executada. Ao terminar, clique em OK e diga ‘prossiga’.”**.

Assim que esta ação for executada, um novo vértice, contendo o módulo “Play Prompt”, é inserido no fluxo de diálogo, com uma aresta ligando o vértice a que se referia a ação anterior que disparou esta com tal novo vértice. Além disso, a tela de configuração do módulo “Play Prompt” (Figura 3.8) é exibida para o usuário.

Existe somente uma próxima ação possível. Após o usuário realizar a tarefa e dizer “prossiga”, o sistema deve oferecer a ele todas as opções disponíveis para continuar o projeto do diálogo. Isto é feito na pergunta principal do *wizard*.

d) Ação da Pergunta Principal

Transcrição do áudio: **“Em seguida, diga ‘mensagem’ para que o sistema execute nova mensagem; diga ‘pergunta’ para que o sistema realize uma pergunta; ou diga ‘encerrar’ para o sistema encerrar o atendimento do cliente.”**.

Nesta pergunta principal, a locução “em seguida” é utilizada para indicar continuidade na conversa. Além disso, a opção para encerrar o atendimento já é disponibilizada, pois o usuário pode, muito bem, desejar que seu sistema de diálogo execute somente uma mensagem de alerta dizendo que o *call center* está em manutenção e pedindo desculpas pelo transtorno.

Existem três próximas ações possíveis. Caso o usuário diga “mensagem”, isso significa que um módulo “Play Prompt” deve ser inserido no fluxo e, portanto, a próxima ação do *wizard* deve ser de configuração de tal módulo. Caso o usuário diga “pergunta”, o sistema necessita saber se o usuário se refere a uma pergunta simples ou com múltiplas respostas, portanto a próxima ação é realizar um questionamento sobre o tipo de pergunta que o usuário deseja. Caso o usuário diga “encerrar” o sistema deve verificar se existe alguma ação na pilha de retorno e, em caso positivo, executar a ação no topo da pilha; em caso negativo, o sistema deve executar o áudio de encerramento do *wizard*, transcrito a seguir: **“Fim do *wizard*. Seu projeto agora está criado.”**.

A pilha de retorno é utilizada nos casos em que há múltiplas saídas possíveis para um módulo do sistema (caso do módulo de perguntas de múltiplas respostas). Nestes casos, o sistema deve seguir a execução do *wizard* por uma das possíveis saídas, mas deve colocar em uma pilha as ações correspondentes para as demais saídas de modo que possa voltar a elas assim que a execução do *wizard* encerrar no ramo daquela saída. A ideia por trás desse comportamento é projetar o diálogo em profundidade, isto é, seguindo o caminho de uma busca em profundidade no grafo representativo do diálogo.

e) Ação do Tipo de Pergunta

Transcrição do áudio: **“Se desejar que seja uma pergunta simples, diga ‘simples’. Para uma pergunta com múltiplas respostas, diga ‘múltipla’.”**.

Aqui o sistema deseja saber se a pergunta desejada pelo usuário é simples ou com múltiplas respostas.

Existem duas próximas ações possíveis. Caso o usuário diga “simples”, isso significa que um módulo “Simple Question” deve ser inserido no fluxo e, portanto, a próxima ação do *wizard* deve ser de configuração de tal módulo. Caso o usuário diga “múltipla”, o sistema necessita saber se o usuário gostaria de utilizar a pergunta do tipo “sim ou não”, ou, em caso negativo, configurar sua própria pergunta de múltiplas respostas. Portanto a próxima ação neste caso é realizar um questionamento sobre o desejo do usuário de usar uma pergunta do tipo sim ou não.

f) Ação de Configuração do Módulo “Simple Question”

Transcrição do áudio: **“Preencha o formulário com o nome da pergunta; selecione o arquivo que contém a pergunta a ser feita; e preencha a lista de respostas possíveis, separadas por ponto-e-vírgula. Ao terminar, clique em OK e diga ‘prossiga’.”**.

Assim que esta ação for executada, um novo vértice, contendo o módulo “Simple Question”, é inserido no fluxo de diálogo, com uma aresta ligando o vértice a que se referia a ação anterior que disparou esta com tal novo vértice. Além disso, a tela de configuração do módulo “Simple Question” (Figura 3.9) é exibida para o usuário.

Existe somente uma próxima ação possível. Após o usuário realizar a tarefa e dizer “prossiga”, o sistema deve questioná-lo se ele deseja que a resposta do cliente à pergunta simples seja registrada em log.

g) Ação do Registro em Log

Transcrição do áudio: **“Você gostaria de registrar em log a resposta do cliente.”**.

Existem duas próximas ações possíveis. Caso o usuário diga “sim”, um novo vértice, contendo o módulo “Log”, é inserido no fluxo de diálogo, com uma aresta ligando o vértice a que se referia a ação anterior que disparou esta com tal novo vértice. Em seguida, o sistema deve oferecer a ele todas as opções disponíveis para continuar o projeto do diálogo, através da ação da pergunta principal do *wizard*. Caso o usuário diga “não”, o sistema não realiza nenhuma tarefa adicional, invocando simplesmente a ação da pergunta principal do *wizard* para oferecer ao usuário projetista todas as opções disponíveis para continuar o projeto.

h) Ação da Opção pela Pergunta do Tipo Sim ou Não

Transcrição do áudio: **“Deseja que a pergunta seja do tipo ‘sim ou não’? Diga ‘sim’ em caso positivo, e ‘não’ caso contrário.”**.

Aqui o sistema deseja saber se a pergunta de múltiplas respostas que o usuário deseja é do tipo “sim ou não”.

Existem duas próximas ações possíveis. Caso o usuário diga “sim”, isso significa que um módulo “Yes/No Question” deve ser inserido no fluxo e, portanto, a próxima ação do *wizard* deve ser de configuração de tal módulo. Caso o usuário diga “não”, isso significa que um módulo “Multiple Question” deve ser inserido no fluxo e, portanto, a próxima ação do *wizard* deve ser de configuração de tal módulo.

i) Ação de Configuração do Módulo “Yes/No Question”

Transcrição do áudio: **“Preencha o formulário com o nome da pergunta e selecione o arquivo que contém a pergunta a ser feita. Ao terminar, clique em OK e diga ‘prossiga’.”**.

Assim que esta ação for executada, um novo vértice, contendo o módulo “Yes/No Question”, é inserido no fluxo de diálogo, com uma aresta ligando o vértice

a que se referia a ação anterior que disparou esta com tal novo vértice. Além disso, a tela de configuração do módulo “Yes/No Question” (Figura 3.12) é exibida para o usuário.

Existe somente uma próxima ação possível. Após o usuário realizar a tarefa e dizer “prossiga”, o sistema deve oferecer a ele todas as opções disponíveis para continuar o projeto do diálogo para o caso de a resposta do cliente ser “sim”. Além disso, o sistema deve incluir na pilha de retorno a mesma ação para o caso de a resposta do cliente ser “não”.

j) Ação de Tratamento de Resposta para “Yes/No Question”

Transcrição do áudio: **“Na pergunta do tipo “sim ou não”, o que o sistema deve fazer para uma resposta ‘sim’ (‘não’)? Diga ‘mensagem’ para que o sistema execute uma mensagem; diga ‘pergunta’ para que o sistema realize uma pergunta; ou diga ‘encerrar’ para o sistema encerrar o atendimento ao cliente.”**.

Dependendo da resposta anterior do cliente sendo considerada, o áudio muda, como indicado na transcrição. Ademais, todas as opções disponíveis são oferecidas ao usuário como na pergunta principal do *wizard*, assim como as próximas ações possíveis são as mesmas da ação da pergunta principal.

k) Ação de Configuração do Módulo “Multiple Question”

Transcrição do áudio: **“Preencha o formulário com o nome da pergunta; selecione o arquivo que contém a pergunta a ser feita; selecione o número de opções de resposta e informe-as em seguida. Ao terminar, clique em OK e diga ‘prossiga’.”**.

Assim que esta ação for executada, um novo vértice, contendo o módulo “Multiple Question”, é inserido no fluxo de diálogo, com uma aresta ligando o vértice a que se referia a ação anterior que disparou esta com tal novo vértice. Além disso, a tela de configuração do módulo “Multiple Question” (Figura 3.10) é exibida para o usuário.

Existe somente uma próxima ação possível. Após o usuário realizar a tarefa e dizer “prossiga”, o sistema deve oferecer a ele todas as opções disponíveis para continuar o projeto do diálogo para o caso de a resposta do cliente ser a primeira

informada. Além disso, o sistema deve incluir na pilha de retorno a mesma ação para cada uma das demais respostas informadas.

l) Ação de Tratamento de Resposta para “Multiple Question”

Transcrição do áudio: **“Na pergunta do tipo ‘múltiplas respostas’ mostrada em sua tela, o que o sistema deve fazer para a primeira (segunda, terceira, etc.) opção de resposta, apresentada em destaque? Diga ‘mensagem’ para que o sistema execute uma mensagem; diga ‘pergunta’ para que o sistema realize uma pergunta; ou diga ‘encerrar’ para o sistema encerrar o atendimento ao cliente.”**.

Dependendo da resposta anterior do cliente sendo considerada, o áudio muda, como indicado na transcrição. Além disso, sempre que essa ação for executada, uma tela somente para leitura será exibida para o usuário, contendo o nome da pergunta de múltiplas respostas em questão e a lista de opções de respostas configuradas para ela, sendo que a resposta para a qual o usuário deve decidir a próxima ação aparece destacada, como mostrado na Figura 3.14.

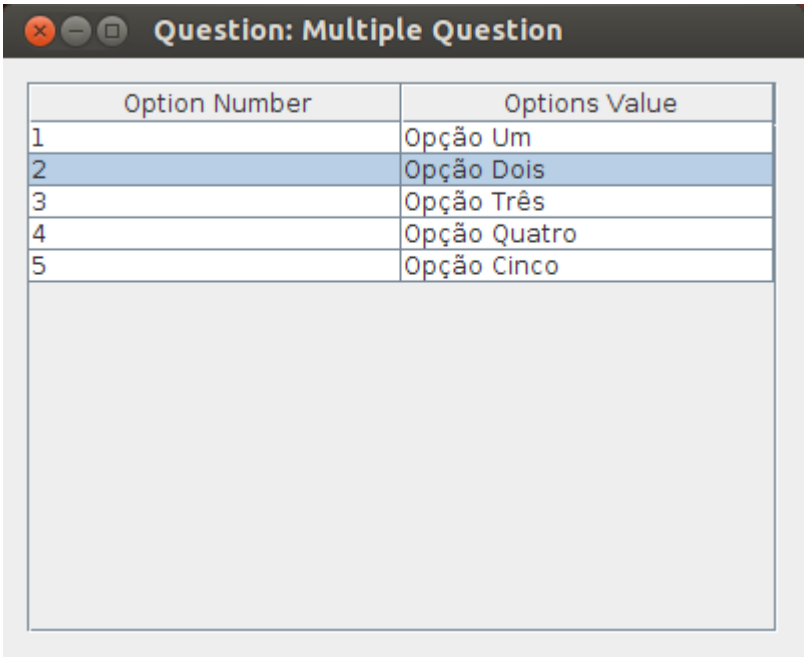


Figura 3.14 – Tela para visualização da resposta de pergunta com múltiplas respostas sendo considerada pela ação do *wizard*.

Ademais, todas as opções disponíveis são oferecidas ao usuário como na pergunta principal do *wizard*, assim como as próximas ações possíveis são as mesmas da ação da pergunta principal.

As Figuras 3.15 a 3.21 mostram esquemas do fluxo de diálogo do *wizard*, explicado anteriormente. Nas figuras, as caixas na cor roxas são indicativas somente do início de cada subfluxo; as caixas na cor azul representam as ações apresentadas para o *wizard*; e as caixas na cor marrom representam subfluxos.

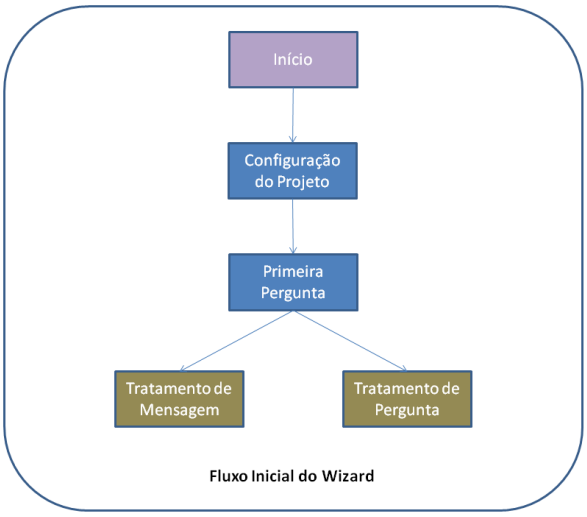


Figura 3.15 – Esquema do subfluxo de início do *wizard*.

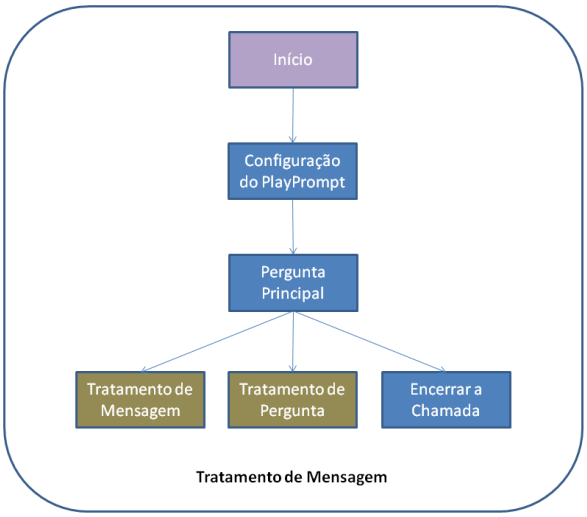


Figura 3.16 – Esquema do subfluxo para tratamento de mensagem.

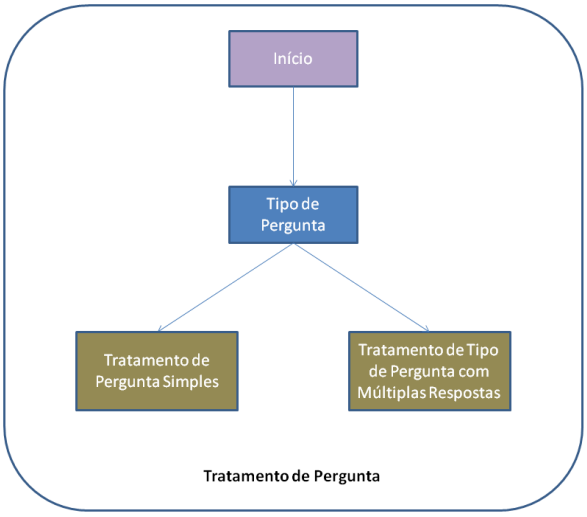


Figura 3.17 – Esquema do subfluxo para tratamento de pergunta.

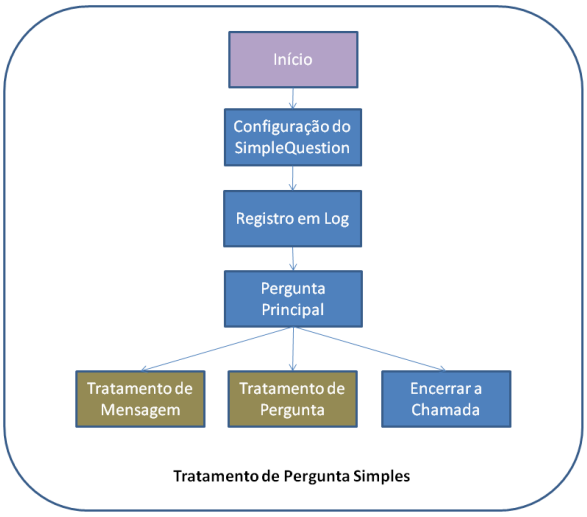


Figura 3.18 – Esquema do subfluxo para tratamento de pergunta simples.

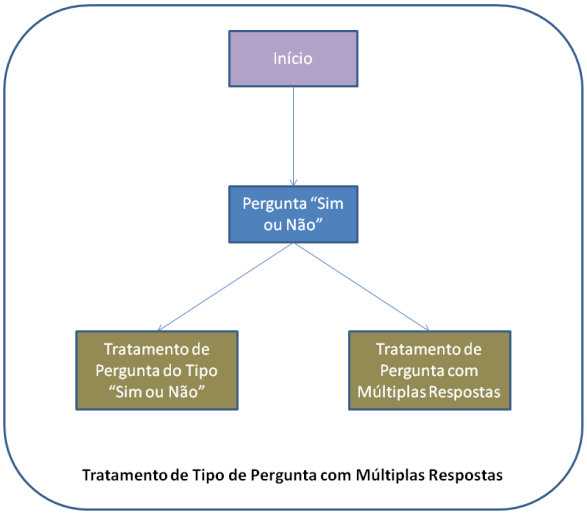


Figura 3.19 – Esquema do subfluxo de tratamento para tipo de pergunta com múltiplas respostas.

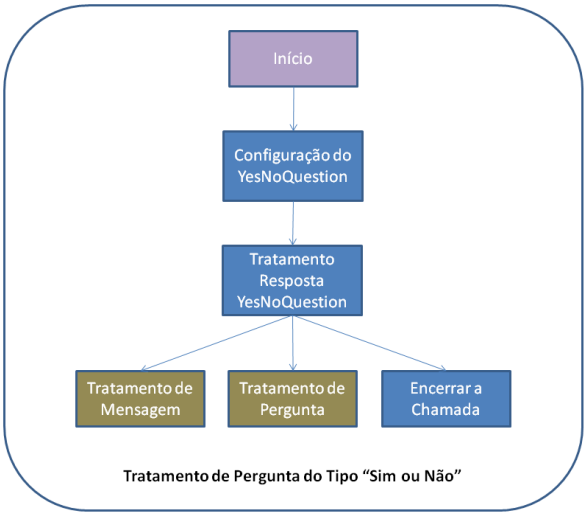


Figura 3.20 – Esquema do subfluxo de tratamento para pergunta do tipo "sim ou não".

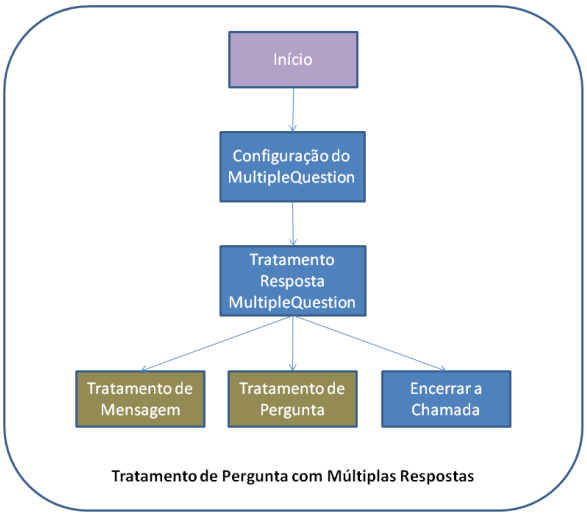


Figura 3.21 – Esquema do subfluxo de tratamento para pergunta com múltiplas respostas.

3.3.6 Menu "Help"

Há duas funcionalidades disponíveis na opção "Help" do menu principal. Elas são funcionalidades somente informativas para o usuário. A primeira delas ("About") exibe informações gerais sobre o sistema DialogBuilder: sua versão atual, um resumo do que ele é e do seu propósito e um histórico de seu desenvolvimento.

A segunda funcionalidade (“Tutorial”) exibe um pequeno roteiro de como o usuário deve proceder para, manualmente, implantar o sistema de diálogo no Asterisk usando os insumos gerados pelo DialogBuilder, conforme discutido na seção 3.3.4.2.

3.4 VALIDAÇÃO DO SISTEMA

O sistema foi validado através da criação, tanto pela forma tradicional quanto através do *wizard*, de projetos de sistema de diálogo de testes, em níveis crescentes de complexidade, à medida que os módulos iam sendo desenvolvidos e integrados aos demais. Além disso, o *wizard*, como um sistema de diálogo, foi avaliado individualmente, conforme detalhado na seção 3.4.3.

Cada projeto de sistema de diálogo criado para validação do DialogBuilder era exportado para o Asterisk e implantado em uma instância deste. Após isso, eram gerados pequenos casos de teste para o sistema de diálogo em questão e, com base em tais casos, os testes eram executados por um usuário que realizava chamadas para o Asterisk através de um *software* VoIP chamado Twinkle[28]. A ideia dos testes era garantir que o sistema de diálogo criado rodasse tal qual foi projetado e, por isso, os casos de testes buscavam cobrir as possíveis ramificações no fluxo do diálogo.

Os testes foram todos bem sucedidos, isto é, os casos de testes foram executados e o sistema de diálogo projetado se comportou conforme esperado. Para evitar ser prolixo, somente serão apresentados, a seguir, dois dos mais complexos casos considerados: um para projeto tradicional e outro para *wizard*.

[28] http://www.twinklephone.com. Acesso em: 07 dez. 2013.

3.4.1 Projeto Tradicional

Este exemplo trata de um sistema de registro de ponto. Os funcionários de uma pequena empresa devem registrar seus pontos através do *call center* interno da corporação. Basicamente, o usuário deve ligar para o ramal 100 e, após ouvir uma mensagem de saudação, escolher se deseja registrar seu ponto ou ouvir um pequeno tutorial de como funciona o sistema de registro de ponto da empresa. O registro de ponto ocorre através do registro nos arquivos de *log* do servidor do nome do funcionário (o *log* já traz o dia e a hora da chamada).

Para fazer o projeto deste sistema, optou-se por utilizar a forma tradicional através da interface gráfica do DialogBuilder. Primeiramente, os seguintes arquivos de áudio foram gravados, junto com suas respectivas transcrições:

a) Mensagem de Saudação (saudacao.wav): **“Olá, seja bem-vindo ao nosso sistema de registro de ponto.”**;
b) Mensagem do Menu de Opções (menu.wav): **“Diga ‘registro’ para realizar seu registro de ponto ou diga ‘tutorial’ para ouvir uma explicação sobre como funciona esse sistema.”**;
c) Identificação do Funcionário (nome.wav): **“Para proceder com o registro, responda: qual o seu nome?”**;
d) Tutorial (tutorial.wav): o tutorial não será transcrito para economia de espaço;
e) Mensagem de Confirmação de Registro (confirmacao.wav): **“Obrigado! O registro de seu ponto foi efetuado com sucesso.”**.

Em seguida, o fluxo do sistema foi desenhado como mostrado na Figura 3.22. O módulo utilizado para a “Saudação” foi o “Play Prompt”, onde foi utilizado como *prompt* o arquivo “saudacao.wav”. O “Menu” é um módulo “Multiple Question”, com o arquivo “menu.wav” como áudio da pergunta e com duas opções de resposta: “registro” e “tutorial”. O “Nome” é um módulo “Simple Question”, com o arquivo “nome.wav” como arquivo de áudio. Nas possíveis opções de resposta, os nomes de todos os funcionários da empresa devem ser listados, sendo que para este exemplo foram considerados os nomes “João”, “Caio”, “Mateus”, “Lucas”, “Alberto”,

“Alexandre”, “Bernardo”, “Carlos” e “Davi”. O módulo “Log” foi utilizado para registrar o nome do funcionário que registrou seu ponto. Tanto o “Tutorial” quanto o “Confirmação” são módulos “Play Prompt” com os arquivos “tutorial.wav” e “confirmacao.wav”, respectivamente, como arquivos de *prompt*. E, por último, o módulo “Flow End” encerra a ligação.

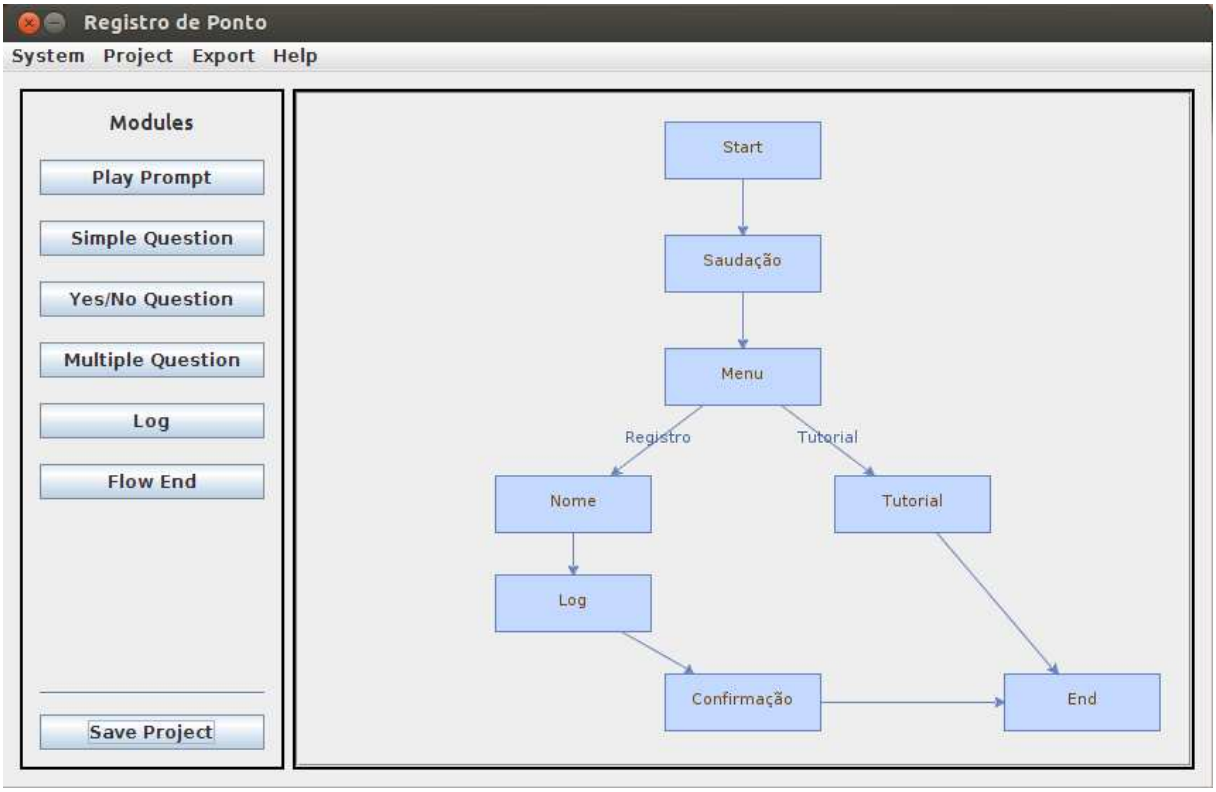


Figura 3.22 – Sistema de diálogo “Registro de Ponto”.

Como resultado da exportação deste sistema, além dos arquivos de áudio já descritos, foram geradas duas gramáticas, “Menu” e “Nome”, sendo que a primeira com duas regras: uma para a palavra “Registro” e outra para a palavra “Tutorial”; e a segunda com nove regras, uma para nome considerado, conforme já mencionado. Por último, foi gerado o *script dialplan* do sistema, mostrado a seguir:

```
[default]
exten => 100,1,answer()
exten => 100,2,playback(saudacao)
exten => 100,3,playback(menu)
exten => 100,4,
      EAGI(adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav)
exten => 100,5,Set(CONF=${RECRET:0})
exten => 100,6,GotoIf($[${CONF} = Registro]?8:7)
exten => 100,7,GotoIf($[${CONF} = Tutorial]?14:8)
```

```
exten => 100,8,playback(nome)
exten => 100,9,
      EAGI(adintool.py,julius.jconf,Nome,6,1,,/tmp/audio.wav)
exten => 100,10,Set(ANSWER=${RECRET:0})
exten => 100,11,log(NOTICE,${ANSWER})
exten => 100,12,playback(confirmacao)
exten => 100,13,hangup()
exten => 100,14,playback(tutorial)
exten => 100,15,hangup()
```

Dois casos de testes foram utilizados para validação deste sistema de diálogo: no primeiro, o tutorial é solicitado; no segundo, é realizado o registro do ponto do funcionário Alberto. A Figura 3.23 mostra o registro de execução, feito pelo Asterisk, do primeiro caso de teste, enquanto a Figura 3.24 mostra o registro de execução do segundo caso. Em ambos, os trechos mais importantes foram destacados.

```
  == Using SIP RTP CoS mark 5
    -- Executing [100@default:1] Answer("SIP/5000-00000002", "") in new stack
    -- Executing [100@default:2] Playback("SIP/5000-00000002", "saudacao") in new stack
    -- <SIP/5000-00000002> Playing 'saudacao.slin' (language 'pt')
    -- Executing [100@default:3] Playback("SIP/5000-00000002", "menu") in new stack
    -- <SIP/5000-00000002> Playing 'menu.slin' (language 'pt')
    -- Executing [100@default:4] EAGI("SIP/5000-00000002",
"adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav") in new stack
    -- Launched AGI Script /var/lib/asterisk/agi-bin/adintool.py
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: jconf /usr/src/coruja-
asterisk/julius.jconf
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: grammar /usr/src/coruja-
asterisk/grammars/Menu
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: Initiaing julius
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: Waiting for Julius log in /tmp/audio.wav
    -- Playing 'beep' (escape_digits=) (sample_offset 0)
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: Julius finished (<s> tutorial </s>)  1.000
1.000 1.000
    -- <SIP/5000-00000002>AGI Script adintool.py completed, returning 0
    -- Executing [100@default:5] Set("SIP/5000-00000002", "CONF=Tutorial") in new stack
    -- Executing [100@default:6] GotoIf("SIP/5000-00000002", "0?8:7") in new stack
    -- Goto (default,100,7)
    -- Executing [100@default:7] GotoIf("SIP/5000-00000002", "1?14:8") in new stack
    -- Goto (default,100,14)
    -- Executing [100@default:14] Playback("SIP/5000-00000002", "tutorial") in new stack
    -- <SIP/5000-00000002> Playing 'tutorial.slin' (language 'pt')
    -- Executing [100@default:15] Hangup("SIP/5000-00000002", "") in new stack
  == Spawn extension (default, 100, 15) exited non-zero on 'SIP/5000-00000002'
```

Figura 3.23 – Registro de execução do primeiro caso de teste do sistema “Registro de Ponto”.

```
  == Using SIP RTP CoS mark 5
    -- Executing [100@default:1] Answer("SIP/5000-00000004", "") in new stack
    -- Executing [100@default:2] Playback("SIP/5000-00000004", "saudacao") in new stack
    -- <SIP/5000-00000004> Playing 'saudacao.slin' (language 'pt')
    -- Executing [100@default:3] Playback("SIP/5000-00000004", "menu") in new stack
    -- <SIP/5000-00000004> Playing 'menu.slin' (language 'pt')
    -- Executing [100@default:4] EAGI("SIP/5000-00000004",
"adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav") in new stack
    -- Launched AGI Script /var/lib/asterisk/agi-bin/adintool.py
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: jconf /usr/src/coruja-
asterisk/julius.jconf
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: grammar /usr/src/coruja-
asterisk/grammars/Menu
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: Initiaing julius
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: Waiting for Julius log in
/tmp/audio.wav
    -- Playing 'beep' (escape_digits=) (sample_offset 0)
adintool.py,julius.jconf,Menu,6,1,,/tmp/audio.wav: Julius finished (<s> registro </s>)
1.000 1.000 1.000
    -- <SIP/5000-00000004>AGI Script adintool.py completed, returning 0
    -- Executing [100@default:5] Set("SIP/5000-00000004", "CONF=Registro") in new stack
    -- Executing [100@default:6] GotoIf("SIP/5000-00000004", "1?8:7") in new stack
    -- Goto (default,100,8)
    -- Executing [100@default:8] Playback("SIP/5000-00000004", "nome") in new stack
    -- <SIP/5000-00000004> Playing 'nome.slin' (language 'pt')
    -- Executing [100@default:9] EAGI("SIP/5000-00000004",
"adintool.py,julius.jconf,Nome,6,1,,/tmp/audio.wav") in new stack
    -- Launched AGI Script /var/lib/asterisk/agi-bin/adintool.py
adintool.py,julius.jconf,Nome,6,1,,/tmp/audio.wav: jconf /usr/src/coruja-
asterisk/julius.jconf
adintool.py,julius.jconf,Nome,6,1,,/tmp/audio.wav: grammar /usr/src/coruja-
asterisk/grammars/Nome
adintool.py,julius.jconf,Nome,6,1,,/tmp/audio.wav: Initiaing julius
adintool.py,julius.jconf,Nome,6,1,,/tmp/audio.wav: Waiting for Julius log in
/tmp/audio.wav
    -- Playing 'beep' (escape_digits=) (sample_offset 0)
adintool.py,julius.jconf,Nome,6,1,,/tmp/audio.wav: Julius finished (<s> alberto </s>)
1.000 1.000 1.000
    -- <SIP/5000-00000004>AGI Script adintool.py completed, returning 0
    -- Executing [100@default:10] Set("SIP/5000-00000004", "ANSWER=Alberto") in new
stack
    -- Executing [100@default:11] Log("SIP/5000-00000004", "NOTICE,Alberto") in new
stack
[Dec  7 22:15:11] NOTICE[3765][C-00000004]: Ext. 100:11 @ default: Alberto
    -- Executing [100@default:12] Playback("SIP/5000-00000004", "confirmacao") in new
stack
    -- <SIP/5000-00000004> Playing 'confirmacao.slin' (language 'pt')
    -- Executing [100@default:13] Hangup("SIP/5000-00000004", "") in new stack
  == Spawn extension (default, 100, 13) exited non-zero on 'SIP/5000-00000004'
```

Figura 3.24 – Registro de execução do segundo caso de teste do sistema “Registro de Ponto”.

3.4.2 Projeto com *Wizard*

Este exemplo trata de um sistema de avaliação de atendimento, comumente utilizados ao fim de um atendimento telefônico. O cliente é redirecionado para a extensão deste sistema (no exemplo, extensão 200) e, primeiramente, é questionado sobre qual nota daria para o atendimento. Após o registro da nota, o

sistema então pergunta se o cliente recomendaria o *call center* para outras pessoas e então toca uma mensagem de agradecimento, caso a resposta seja positiva; ou uma mensagem se desculpando e prometendo melhorar, caso ela seja negativa. A chamada é encerrada em seguida.

Para fazer o projeto deste sistema, optou-se por utilizar o *wizard*. Primeiramente, os seguintes arquivos de áudio foram gravados, junto com suas respectivas transcrições:

a) Pergunta sobre Nota (nota.wav): **“Olá, que nota, de um a dez, você daria para o atendimento que recebeu?”**;
b) Pergunta sobre Recomendação (recomendacao.wav): **“Obrigado. Você recomendaria nosso *call center* para outras pessoas?”**;
c) Mensagem de Agradecimento (agradecimento.wav): **“Muito obrigado pela sua avaliação.”**;
d) Mensagem de Desculpas (desculpas.wav): **“Pedimos desculpas por qualquer problema e estaremos constantemente trabalhando para melhor atendê-lo. Obrigado pela colaboração.”**.

Em seguida, o *wizard* foi iniciado. A seguir, pode-se ler a transcrição do diálogo com o DialogBuilder e na Figura 3.25, pode-se ver o fluxo de diálogo gerado após a finalização do *wizard*.

[DialogBuilder] **Antes de iniciar o wizard, preencha o formulário com as informações gerais do diálogo. Ao terminar, clique em OK e diga ‘prossiga’.**

[Usuário preenche o formulário com as informações gerais do projeto.]

[Usuário] **Prossiga.**

[DialogBuilder] **Olá! O que você gostaria que o sistema fizesse ao iniciar um atendimento para seu cliente? Diga ‘mensagem’ para o sistema executar uma mensagem de boas-vindas; ou diga ‘pergunta’ para o sistema realizar uma pergunta.**

[Usuário] **Pergunta.**

[DialogBuilder] **Se desejar que seja uma pergunta simples, diga ‘simples’. Para uma pergunta com múltiplas respostas, diga ‘múltipla’.**

[Usuário] **Simples.**

[DialogBuilder] **Preencha o formulário com o nome da pergunta; selecione o arquivo que contém a pergunta a ser feita; e preencha a lista de respostas possíveis, separadas por ponto-e-vírgula. Ao terminar, clique em OK e diga 'prossiga'.**

[Usuário preenche o formulário, selecionando o arquivo "nota.wav" e incluindo como opções de respostas: um, dois, três, quatro, cinco, seis, sete, oito, nove, dez.]

[Usuário] **Prossiga.**

[DialogBuilder] **Você gostaria de registrar em log a resposta do cliente.**

[Usuário] **Sim.**

[DialogBuilder] **Em seguida, diga 'mensagem' para que o sistema execute nova mensagem; diga 'pergunta' para que o sistema realize uma pergunta; ou diga 'encerrar' para o sistema encerrar o atendimento do cliente.**

[Usuário] **Pergunta.**

[DialogBuilder] **Se desejar que seja uma pergunta simples, diga 'simples'. Para uma pergunta com múltiplas respostas, diga 'múltipla'.**

[Usuário] **Múltipla.**

[DialogBuilder] **Deseja que a pergunta seja do tipo 'sim ou não'? Diga 'sim' em caso positivo, e 'não' caso contrário.**

[Usuário] **Sim.**

[DialogBuilder] **Preencha o formulário com o nome da pergunta e selecione o arquivo que contém a pergunta a ser feita. Ao terminar, clique em OK e diga 'prossiga'.**

[Usuário preenche o formulário, selecionando o arquivo "recomendacao.wav"]

[Usuário] **Prossiga.**

[DialogBuilder] **Na pergunta do tipo "sim ou não", o que o sistema deve fazer para uma resposta 'sim'? Diga 'mensagem' para que o sistema execute**

uma mensagem; diga ‘pergunta’ para que o sistema realize uma pergunta; ou diga ‘encerrar’ para o sistema encerrar o atendimento ao cliente.

[Usuário] **Mensagem.**

[DialogBuilder] **Preencha o formulário com o nome da mensagem e selecione o arquivo que contém a mensagem a ser executada. Ao terminar, clique em OK e diga ‘prossiga’.**

[Usuário preenche o formulário, selecionando o arquivo “agradecimento.wav”]

[Usuário] **Prossiga.**

[DialogBuilder] **Em seguida, diga ‘mensagem’ para que o sistema execute nova mensagem; diga ‘pergunta’ para que o sistema realize uma pergunta; ou diga ‘encerrar’ para o sistema encerrar o atendimento do cliente.**

[Usuário] **Encerrar.**

[DialogBuilder] **Na pergunta do tipo “sim ou não”, o que o sistema deve fazer para uma resposta ‘não’? Diga ‘mensagem’ para que o sistema execute uma mensagem; diga ‘pergunta’ para que o sistema realize uma pergunta; ou diga ‘encerrar’ para o sistema encerrar o atendimento ao cliente.**

[Usuário] **Mensagem.**

[DialogBuilder] **Preencha o formulário com o nome da mensagem e selecione o arquivo que contém a mensagem a ser executada. Ao terminar, clique em OK e diga ‘prossiga’.**

[Usuário preenche o formulário, selecionando o arquivo “desculpas.wav”]

[Usuário] **Prossiga.**

[DialogBuilder] **Em seguida, diga ‘mensagem’ para que o sistema execute nova mensagem; diga ‘pergunta’ para que o sistema realize uma pergunta; ou diga ‘encerrar’ para o sistema encerrar o atendimento do cliente.**

[Usuário] **Encerrar.**

[DialogBuilder] **Fim do wizard. Seu projeto agora está criado.**

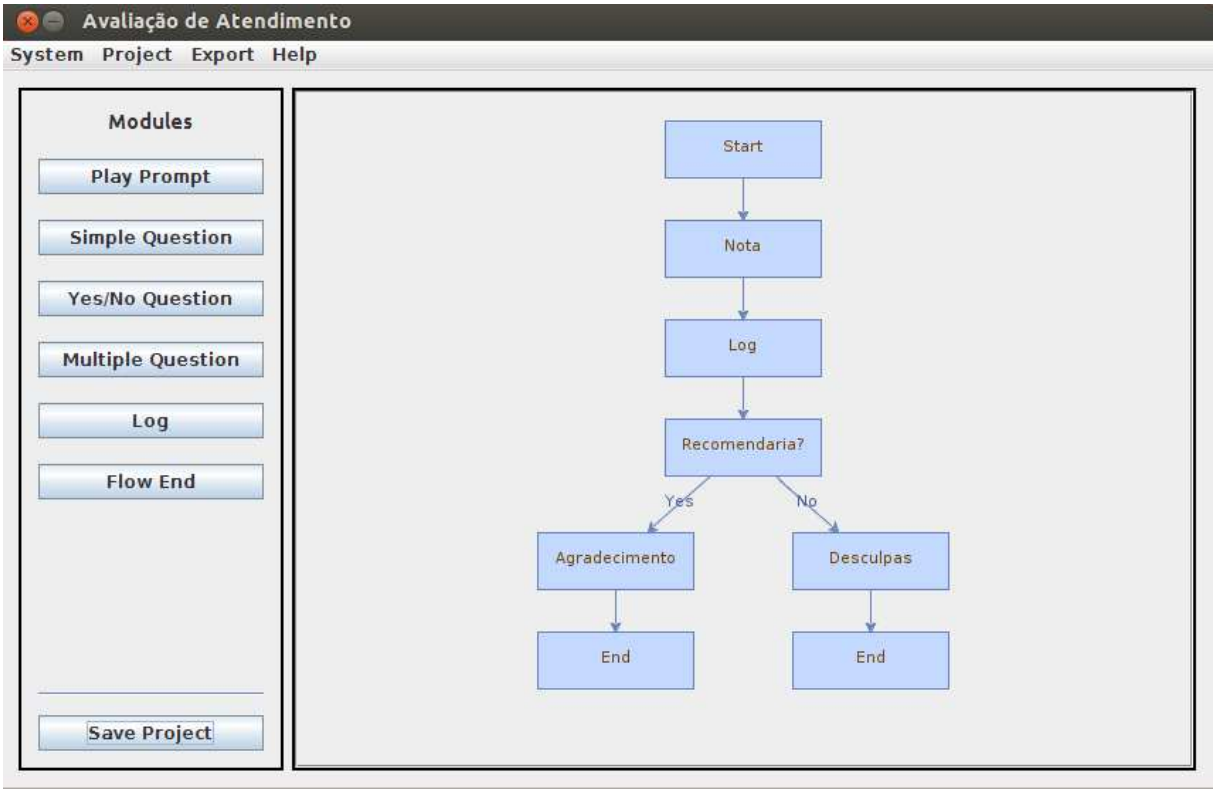


Figura 3.25 – Sistema de diálogo “Avaliação de Atendimento”.

Como resultado da exportação deste sistema, além dos arquivos de áudio já descritos, foram geradas duas gramáticas, “Nota” e “YesNoGrammar”, sendo que a primeira com dez regras: uma para cada numeral de um a dez; e a segunda igual a apresentada na seção 2.3.4.1. Por último, foi gerado o *script dialplan* do sistema, mostrado a seguir:

```
[default]
exten => 200,1,answer()
exten => 200,2,playback(nota)
exten => 200,3,
      EAGI(adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav)
exten => 200,4,Set(ANSWER=${RECRET:0})
exten => 200,5,log(NOTICE,${ANSWER})
exten => 200,6,playback(recomendacao)
exten => 200,7,
      EAGI(adintool.py,julius.jconf,YesNoGrammar,6,1,,
           /tmp/audio.wav)
exten => 200,8,Set(CONF=${RECRET:12:1})
exten => 200,9,GotoIf($[${CONF} = 1]?10:12)
exten => 200,10,playback(agradecimento)
exten => 200,11,hangup()
```

```
exten => 200,12,playback(desculpas)
exten => 200,13,hangup()
```

Dois casos de testes foram utilizados para validação deste sistema de diálogo: no primeiro, o usuário deu nota nove para o atendimento e disse que recomendaria o *call center*; no segundo, a nota foi quatro e não haveria recomendação. A Figura 3.26 mostra o registro de execução, feito pelo Asterisk, do primeiro caso de teste, enquanto a Figura 3.27 mostra o registro de execução do segundo caso. Em ambos, os trechos mais importantes foram destacados.

```
  == Using SIP RTP CoS mark 5
    -- Executing [200@default:1] Answer("SIP/5000-0000000c", "") in new stack
    -- Executing [200@default:2] Playback("SIP/5000-0000000c", "nota") in new stack
    -- <SIP/5000-0000000c> Playing 'nota.slin' (language 'pt')
    -- Executing [200@default:3] EAGI("SIP/5000-0000000c",
"adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav") in new stack
    -- Launched AGI Script /var/lib/asterisk/agi-bin/adintool.py
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: jconf /usr/src/coruja-
asterisk/julius.jconf
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: grammar /usr/src/coruja-
asterisk/grammars/Nota
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: Initiaing julius
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: Waiting for Julius log in
/tmp/audio.wav
    -- Playing 'beep' (escape_digits=) (sample_offset 0)
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: Julius finished (<s> nove </s>)
1.000 1.000 1.000
    -- <SIP/5000-0000000c>AGI Script adintool.py completed, returning 0
    -- Executing [200@default:4] Set("SIP/5000-0000000c", "ANSWER=nove") in new
stack
    -- Executing [200@default:5] Log("SIP/5000-0000000c", "NOTICE,nove") in new
stack
[Dec  7 22:39:07] NOTICE[4099][C-0000000c]: Ext. 200:5 @ default: nove
    -- Executing [200@default:6] Playback("SIP/5000-0000000c", "recomendacao") in
new stack
    -- <SIP/5000-0000000c> Playing 'recomendacao.slin' (language 'pt')
    -- Executing [200@default:7] EAGI("SIP/5000-0000000c",
"adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav") in new stack
    -- Launched AGI Script /var/lib/asterisk/agi-bin/adintool.py
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: jconf /usr/src/coruja-
asterisk/julius.jconf
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: grammar /usr/src/coruja-
asterisk/grammars/YesNoGrammar
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: Initiaing julius
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: Waiting for Julius log in
/tmp/audio.wav
    -- Playing 'beep' (escape_digits=) (sample_offset 0)
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: Julius finished (<s> S_i~
</s>)  1.000 0.780 1.000
    -- <SIP/5000-0000000c>AGI Script adintool.py completed, returning 0
    -- Executing [200@default:8] Set("SIP/5000-0000000c", "CONF=1") in new stack
    -- Executing [200@default:9] GotoIf("SIP/5000-0000000c", "1?10:12") in new stack
    -- Goto (default,200,10)
    -- Executing [200@default:10] Playback("SIP/5000-0000000c", "agradecimento") in
new stack
    -- <SIP/5000-0000000c> Playing 'agradecimento.slin' (language 'pt')
    -- Executing [200@default:11] Hangup("SIP/5000-0000000c", "") in new stack
  == Spawn extension (default, 200, 11) exited non-zero on 'SIP/5000-0000000c'
```

Figura 3.26 – Registro de execução do primeiro caso de teste do sistema “Avaliação de Atendimento”.

```
  == Using SIP RTP CoS mark 5
    -- Executing [200@default:1] Answer("SIP/5000-0000000d", "") in new stack
    -- Executing [200@default:2] Playback("SIP/5000-0000000d", "nota") in new stack
    -- <SIP/5000-0000000d> Playing 'nota.slin' (language 'pt')
    -- Executing [200@default:3] EAGI("SIP/5000-0000000d",
"adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav") in new stack
    -- Launched AGI Script /var/lib/asterisk/agi-bin/adintool.py
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: jconf /usr/src/coruja-
asterisk/julius.jconf
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: grammar /usr/src/coruja-
asterisk/grammars/Nota
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: Initiaing julius
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: Waiting for Julius log in
/tmp/audio.wav
    -- Playing 'beep' (escape_digits=) (sample_offset 0)
adintool.py,julius.jconf,Nota,6,1,,/tmp/audio.wav: Julius finished (<s> quatro </s>)
1.000 1.000 1.000
    -- <SIP/5000-0000000d>AGI Script adintool.py completed, returning 0
    -- Executing [200@default:4] Set("SIP/5000-0000000d", "ANSWER=quatro") in new stack
    -- Executing [200@default:5] Log("SIP/5000-0000000d", "NOTICE,quatro") in new stack
[Dec  7 22:40:54] NOTICE[4149][C-0000000d]: Ext. 200:5 @ default: quatro
    -- Executing [200@default:6] Playback("SIP/5000-0000000d", "recomendacao") in new
stack
    -- <SIP/5000-0000000d> Playing 'recomendacao.slin' (language 'pt')
    -- Executing [200@default:7] EAGI("SIP/5000-0000000d",
"adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav") in new stack
    -- Launched AGI Script /var/lib/asterisk/agi-bin/adintool.py
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: jconf /usr/src/coruja-
asterisk/julius.jconf
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: grammar /usr/src/coruja-
asterisk/grammars/YesNoGrammar
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: Initiaing julius
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: Waiting for Julius log in
/tmp/audio.wav
    -- Playing 'beep' (escape_digits=) (sample_offset 0)
adintool.py,julius.jconf,YesNoGrammar,6,1,,/tmp/audio.wav: Julius finished (<s> não
</s>)  1.000 0.736 1.000
    -- <SIP/5000-0000000d>AGI Script adintool.py completed, returning 0
    -- Executing [200@default:8] Set("SIP/5000-0000000d", "CONF=0") in new stack
    -- Executing [200@default:9] GotoIf("SIP/5000-0000000d", "0?10:12") in new stack
    -- Goto (default,200,12)
    -- Executing [200@default:12] Playback("SIP/5000-0000000d", "desculpas") in new
stack
    -- <SIP/5000-0000000d> Playing 'desculpas.slin' (language 'pt')
    -- Executing [200@default:13] Hangup("SIP/5000-0000000d", "") in new stack
  == Spawn extension (default, 200, 13) exited non-zero on 'SIP/5000-0000000d'
```

Figura 3.27 – Registro de execução do segundo caso de teste do sistema “Avaliação de Atendimento”.

3.4.3 Avaliação do Sistema de Diálogo do *Wizard*

Para a avaliação do sistema de diálogo utilizado no *wizard*, buscou-se seguir alguns dos critérios tradicionalmente propostos para tal, conforme discutido na seção 2.2. Foi realizado um experimento para avaliar o módulo de reconhecimento de voz do *wizard* e outro experimento para avaliar a usabilidade do sistema de diálogo em si.

O experimento para avaliar o módulo ASR foi: utilizando cada uma das quatro gramáticas geradas para o *wizard*, um usuário repetiu cinco vezes cada uma das palavras que nelas constavam. Além disso, novamente para cada gramática, o usuário proferiu cinco outras palavras que não constavam nela.

Os resultados foram bons: a taxa de acerto das palavras que constavam na gramática foi de 95%, o que significa que o módulo está reconhecendo bem o que deve reconhecer. A provável explicação para uma taxa de acertos como a obtida é o número reduzido de palavras em cada gramática. Por outro lado, 75% das palavras que não constavam nas gramáticas foram “reconhecidas” como uma das palavras constantes da gramática. Tal fato significa que o reconhecedor está gerando falsos positivos. Isso não é bom e precisa ser trabalho futuramente, apesar de não impedir o uso do aplicativo, desde que o usuário realize o projeto em um ambiente com barulho controlado e se restrinja a responder às questões e tarefas do *wizard* da forma correta.

Para avaliar o sistema de diálogo do *wizard*, solicitou-se a um usuário sem experiência prévia com Asterisk ou linguagem *dialplan* que projetasse o sistema de diálogo “Avaliação de Atendimento”, mostrado na seção 3.4.2. Os requisitos do sistema foram explicados ao projetista e nenhuma instrução prévia sobre o uso do *wizard* foi repassada a ele. A seguir, as observações feitas durante o experimento são analisadas.

Primeiramente, houve dificuldade de o projetista entender o significado do campo “contexto” nas informações gerais do projeto. De fato, a informação “contexto” é muito particular do Asterisk e confunde inicialmente. Ele precisa ser mais bem explicado durante o *wizard* para que o usuário o entenda.

Em segundo lugar, houve problema com a velocidade de resposta do projetista em alguns momentos. Em certos casos, quando a resposta era feita muito rapidamente, o sistema tentava executar as ações relativas à resposta do projetista antes de finalizar ações anteriores, resultando em erros. Um ponto a ser melhorado é não deixar o sistema tratar uma resposta do projetista antes de finalizar as ações que devam ser executadas em primeiro lugar. Este problema fez com que o projetista precisasse reiniciar o *wizard* duas vezes.

Foi notado, também, que houve certa confusão do projetista com a diferença entre a pergunta simples e as perguntas com múltiplas respostas. O motivo da confusão é o fato de que a pergunta é simples não porque não tenha opções de resposta, mas sim porque não gera um fluxo diferente para cada uma de tais opções. É importante buscar formas de deixar isso mais claro, seja em manuais ou tutoriais sobre o uso do *wizard*.

Durante o experimento, o projetista ainda precisou reiniciar mais uma vez sua execução quando teve dificuldades para preencher a tela de configuração de pergunta simples. As dificuldades surgiram quando o projetista não prestou atenção às instruções do *wizard* para ir logo ao preenchimento dos campos. Um ponto de melhoria é apresentar a tela somente após as instruções terem sido ditas.

Na quarta tentativa de execução, o projetista já não teve dificuldades para terminar o projeto do sistema de diálogos.

O resultado final foi satisfatório, principalmente porque o usuário não recebeu qualquer instrução sobre como proceder durante o *wizard*, aprendendo na prática como utilizá-lo.

Por último, questionou-se ao projetista qual sua impressão geral da ferramenta e sua opinião foi de que ela é bastante útil no projeto de sistemas para *call center* e também é fácil de usar, desde que não se tente acelerar demasiadamente o processo do *wizard*, o que acaba gerando erros. Além disso, ele sentiu falta de formas para voltar atrás e mudar uma decisão que tenha tomado de maneira equivocada, sem precisar ter que reiniciar todo o projeto. Este último ponto, de fato, já era de conhecimento e é um ponto que precisa ser abordado futuramente.

4 CONCLUSÕES E SUGESTÕES PARA TRABALHOS FUTUROS

De acordo com o exposto nos capítulos 2 e 3, pode-se concluir que aquilo que se propôs para este trabalho foi realizado, isto é, desenvolveu-se um aplicativo que torna possível que o próprio usuário possa projetar e construir um sistema de diálogo para *call center* baseado na arquitetura do *software* Asterisk, seguindo todas as diretivas de pesquisa definidas na seção 1.3.

Além disso, assim como o *software* MS FrontPage[29] rompeu o paradigma inicial de desenvolvimento de portais HTML, possibilitando ao próprio usuário criar suas páginas, sem depender de empresas com este *know-how*, conclui-se que o aplicativo DialogBuilder busca quebrar o paradigma de que só se é possível dotar um *call center* com uma ferramenta de diálogos com processamento de linguagem natural através de profissionais especializados. Com o DialogBuilder, qualquer empresa poderá projetar e desenvolver diretamente sistemas de diálogos a serem hospedados no *software* Asterisk.

Adicionalmente, mesmo que haja soluções comerciais semelhantes, como discutido na seção 3.1, conclui-se que o DialogBuilder se diferencia pelo *wizard* que disponibiliza, pois este facilita o processo de projeto do sistema de diálogo, em especial para o usuário iniciante.

Pode-se concluir também que o DialogBuilder representa uma real opção técnica e econômica às pequenas empresas para utilização de sistemas de diálogo com poder de processamento de linguagem natural em seus *call centers*.

Há de se atentar, contudo, para a relação de compromisso entre a qualidade do sistema de diálogo projetado e exportado para Asterisk através do aplicativo DialogBuilder e o baixo custo desta alternativa, pois por se tratar de uma geração automática de código para Asterisk, o produto final possui características gerais que se aplicam à maioria dos casos, mas que poderiam ser otimizadas, por um

---

[29] http://www.homehost.com.br/artigos/microsoft_frontpage_historia_e_conceitos-047.html. Acesso em: 01 dez. 2013.

especialista, em casos específicos. Entretanto, tais pequenas melhorias não impedem que se use o DialogBuilder para projetar inúmeras aplicações de interesse.

4.1 SUGESTÕES PARA TRABALHOS FUTUROS

Buscando evoluir o aplicativo DialogBuilder, os seguintes trabalhos são sugeridos:

1. Desenvolver e integrar ao aplicativo DialogBuilder novos módulos que tornem possível agregar mais funcionalidades aos sistemas de diálogos como: módulos de pergunta simples especializados para respostas específicas como números, cartões de crédito, CPF, códigos-postais, datas, entre outras; módulos que possibilitem acesso a bases de dados; módulos que possibilitem envio de e-mails; módulos que possibilitem decisões de fluxo; módulos que possibilitem encaminhamento de chamadas; entre outros módulos que tragam mais poder de processamento de linguagem natural ao sistema de diálogo;

2. Integrar o DialogBuilder com o Asterisk para que o sistema de diálogo projetado possa ser automaticamente implantado no Asterisk configurado para uso;

3. Implementar mecanismos para melhorar a usabilidade do sistema, como excluir módulos inseridos erroneamente no fluxo de diálogo, assim como arestas conectadas de maneira errada. Também é necessário um tratamento de erros tanto no *wizard* quanto no módulo de reconhecimento de voz do DialogBuilder para que o usuário possa voltar atrás em uma escolha incorreta que tenha feito ou possa corrigir algum possível problema de reconhecimento de voz;

4. Tratar os pontos de melhoria levantados durante a validação do *wizard*, conforme discutidos na seção 3.4.3;

5. Internacionalizar de maneira completa o aplicativo DialogBuilder, possibilitando seu uso, tanto via interface, quanto via *wizard*, em qualquer língua cadastrada;

6. Incluir no DialogBuilder a possibilidade de trabalhar com projetos de sistemas de diálogos que usem mais de um contexto e/ou extensões;

7. Dotar o DialogBuilder da capacidade de realizar o processo contrário ao atualmente realizado, isto é, receber um *script dialplan* do Asterisk e gerar o projeto do sistema de diálogo correspondente. Essa funcionalidade seria de extrema importância para manutenções de sistemas legados;

8. Incluir no DialogBuilder funcionalidade para emular o funcionamento do Asterisk de forma que um projeto de sistema de diálogo possa ser testado no próprio aplicativo, sem necessidade de implantá-lo no Asterisk para testes de validação.

REFERÊNCIAS

ABDULLATEEF, A. O; MOKHTAR, S. S. M; YUSOFF, R. Z. The Mediating Effects of First Call Resolution on Call Centers' Performance. **Journal of Database Marketing and Customer Strategy Management**, v.18, n.1, p.16-30, 2011.

AKSIN, Z; ARMONY, M; MEHROTRA, V. The Modern Call Center: A Multi-Disciplinary Perspective on Operations Management Research. **Production and Operations Management**, v.16, n.6, p.665-688, 2007.

ANTONIOL, G; FIUTEM, R; FLOR, R; LAZZARI, G. Radiological Reporting Based on Voice Recognition. **Lecture Notes in Computer Science**, v.753, p.242-253, 1993.

AS-SABER, S. N; HOSSAIN, K. Call Centres and Their Role in E-Governance: A Developing Country Perspective. **The Journal of Community Informatics**, v.4, n.3, 2008.

BATISTA, C; COELHO, T; HAICK, B; NETO, N; KLAUTAU, A. Desenvolvimento e Comparação de Reconhecedores de Fala Embarcados e Distribuídos para Android. In: SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 2013, Fortaleza. **Anais...** Fortaleza: [s.n.], 2013.

BATISTA, P; SILVA, P; NETO, N; KLAUTAU, A. A non-Visual Web-Browsing System Using Speech Recognition for Brazilian Portuguese. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL PROCESSING OF THE PORTUGUESE LANGUAGE, 9th, 2010, Porto Alegre. **Proceedings…** Porto Alegre: [s.n.], 2010.

BATISTA, P; COLEM, W; OLIVEIRA, R; SANTOS, H; ARAÚJO, W; NETO, N; KLAUTAU, A. SpeechOO: Uma Extensão de Ditado para o LibreOffice. In: WORKSHOP DE SOFTWARE LIVRE, 2012, Porto Alegre. **Proceedings…** Porto Alegre: [s.n.], 2012.

BATISTA, P. **Avanços em Reconhecimento de Fala para Português Brasileiro e Aplicações:** Ditado no LibreOffice e Unidade de Resposta Audível com Asterisk. 2013. 79p. Dissertação (Mestrado em Engenharia Elétrica) – Instituto de Tecnologia, Universidade Federal do Pará, Belém, 2013.

BELLMAN, R. **Dynamic Programming**. Princeton University Press, 1957.

BENNINGTON, L; CUMMANE, J; CONN, P. Customer Satisfaction and Call Centers: an Australian Study. **International Journal of Service Industry Management**, v.11, n.2, p.162-173, 2000.

BENYON, D; GAMBACK, B; HANSEN, P; MIVAL, O; WEBB, N. How Was Your Day? Evaluating a Conversational Companion. **IEEE Transactions on Affective Computing**, v.4, n.3, p.299-311, 2013.

BERNSEN, N. O; DYBKJAER, L; MINKER, W. Spoken Dialogue Systems Evaluation. In: DYBKJAER, L (Ed.); HEMSEN, H (Ed.); MINKER, W. (Ed.). **Evaluation of Text and Speech Systems**. 2007. p.187-219.

BOHUS, D; RUDNICKY, A. I. Constructing Accurate Beliefs in Spoken Dialog Systems. In: IEEE WORKSHOP ON AUTOMATIC SPEECH RECOGNITION AND UNDERSTANDING, 2005, San Juan. **Proceedings…** San Juan: IEEE, 2005. p.272-277.

BOHUS, D; RAUX, A; HARRIS, T. K; ESKENAZI, M; RUDNICKY, A. I. Olympus: An Open-Source Framework for Conversational Spoken Language Interface Research. In: WORKSHOP ON BRIDGING THE GAP: ACADEMIC AND INDUSTRIAL RESEARCH IN DIALOG TECHNOLOGIES, 2007, Rochester. **Proceedings…** Rochester: ACL, 2007. p.32-39.

BOLTE, T; FLEISCHMAN, R. Still Struggling to Reduce Call Center Costs without Losing Customers? The Right Technologies Lead the Way Out of the Call Center Dilemma. **SAP Insider**, v.8, n.4, October 2007.

CHEN, Y; BAI, Y; TSAI, C; WANG, J; CHEN, B. Voice-Customizable Text-to-Speech for Intelligent Home-Care System. In: INTERNATIONAL CONFERENCE ON ORANGE TECHNOLOGIES, 2013, Taiwan. **Proceedings…** Taiwan: IEEE, 2013. p.239-242.

CIRIGLIANO, R. J; MONTEIRO, C; BARBOSA, F. L; MORAES, J. A; RESENDE JUNIOR, F. G; COUTO, L. Um Conjunto de 1000 Frases Foneticamente Balanceadas para o Português Brasileiro Obtido Utilizando a Abordagem de Algoritmos Genéticos. In: SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 2005, Campinas. **Anais...** Campinas: [s.n.], 2005. p.110-114.

CLARK, A. (Ed.); FOX, C (Ed.); LAPPIN, S (Ed.). **The Handbook of Computational Linguistics and Natural Language Processing**. Wiley-Blackwell, 2012.

DAVIS, S; MERMELSTEIN, P. Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. **IEEE Transactions on Acoustics, Speech and Signal Processing**, v.28, n.4, p.357-366, 1980.

DENG, L; LI, X. Machine Learning Paradigms for Speech Recognition: An Overview. **IEEE Transactions on Audio, Speech, and Language Processing**, v.21, n.5, p.1060-1089, 2013.

DESHMUKH, N; GANAPATHIRAJU, A; PICONE, J. Hierarchical Search for Large-Vocabulary Conversational Speech Recognition: Working toward a Solution to the Decoding Problem. **IEEE Signal Processing Magazine**, v.16, n.5, p.84-107, 1999.

FORBES-RILEY, K; LITMAN, D. Designing and Evaluating a Wizarded Uncertainty-Adaptive Spoken Dialogue Tutoring System. **Computer Speech and Language**, v.25, n.1, p.105-126, 2011.

FOSLER-LUSSIER, E; HE, Y; JYOTHI, P; PRABHAVALKAR, R. Conditional Random Fields in Speech, Audio, and Language Processing. **Proceedings of the IEEE**, v.101, n.5, p.1054-1075, 2013.

GAMMA, E; HELM, R; JOHNSON, R; VLISSIDES, J. **Design Patterns:** Elements of Reusable Object-Oriented Software. Addison-Wesley Professional, 1994.

GOEL, S; BHATTACHARYA, M. Speech Based Dialog Query System over Asterisk PBX Server. In: INTERNATIONAL CONFERENCE ON SIGNAL PROCESSING SYSTEMS, 2nd, 2010, Dalian. **Proceedings…** Dalian: IEEE, 2010. p.752-756.

GOMEZ, R; NAKAMURA, K; NAKADAI, K. Robustness to Speaker Position in Distant-Talking Automatic Speech Recognition. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 2013, Vancouver. **Proceedings…** Vancouver: IEEE, 2013. p.7034-7038.

HERMANSKY, H; COHEN, J. R; STERN, R. M. Perceptual Properties of Current Speech Recognition Technology. **Proceedings of the IEEE**, v.101, n.9, p.1968-1985, 2013.

HOSN, C; BAPTISTA, L. A; IMBIRIBA, T; KLAUTAU, A. New Resources for Brazilian Portuguese: Results for Grapheme-to-Phoneme and Phone Classification. In: INTERNATIONAL TELECOMMUNICATIONS SYMPOSIUM, 2006, Fortaleza. **Proceedings…** Fortaleza: IEEE, 2006. p.477-482.

HUANG, X; ACERO, A; HON, H. **Spoken Language Processing:** A Guide to Theory, Algorithm and System Development. Prentice Hall, 2001.

HUANG, X; ARIKI, Y; JACK, M. **Hidden Markov Models for Speech Recognition.** Illustrated Edition. Edinburgh University Press, 1991.

HUANG, Y; WU, C; CHAO, Y. Personalized Spectral and Prosody Conversion Using Frame-Based Codeword Distribution and Adaptive CRF. **IEEE Transactions on Audio, Speech, and Language Processing**, v.21, n.1, p.51-62, 2013.

ISEKI, F; SATO, Y; KIM, M. W. VoIP System Based on Asterisk for Enterprise Network. In: INTERNATIONAL CONFERENCE ON ADVANCED COMMUNICATION TECHNOLOGY, 13th, 2011, Seoul. **Proceedings…** Seoul: IEEE, 2011. p.1284-1288.

JABAIAN, B; BESACIER, L; LEFEVRE, F. Comparison and Combination of Lightly Supervised Approaches for Language Portability of a Spoken Language Understanding System. **IEEE Transactions on Audio, Speech, and Language Processing**, v.21, n.3, p.636-648, 2013.

JEVTIC, N; KLAUTAU, A; ORLITSKY, A. Estimated Rank Pruning and Java-Based Speech Recognition. In: IEEE WORKSHOP ON AUTOMATIC SPEECH RECOGNITION AND UNDERSTANDING, 2001, Madonna di Campiglio. **Proceedings…** Madonna di Campiglio: IEEE, 2001. p.401-404.

JURAFSKY, D; MARTIN, J. H. **Speech and Language Processing:** An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. 2nd ed. Pearson Prentice Hall, 2008.

KAMM, C. A., WALKER, M. A. Design and Evaluation of Spoken Dialog Systems. In: IEEE WORKSHOP ON AUTOMATIC SPEECH RECOGNITION AND UNDERSTANDING, 1997, Santa Barbara. **Proceedings…** Santa Barbara: IEEE, 1997. p.11-18.

KAPETANIOS, E; TATAR, D; SACAREA, C. **Natural Language Processing:** Semantic Aspects. CRC Press, 2013.

KARHILA, R; REMES, U; KURIMO, M. HMM-Based Speech Synthesis Adaptation Using Noisy Data: Analysis and Evaluation Methods. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 2013, Vancouver. **Proceedings…** Vancouver: IEEE, 2013. p.6930-6934.

KASTNER, M; STANGL, B. Exploring a Text-to-Speech Feature by Describing Learning Experience, Enjoyment, Learning Styles, and Values – A Basis for Future Studies. In: HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES, 46th, 2013, Wailea. **Proceedings…** Wailea: IEEE, 2013. p.3-12.

LADEFOGED, P. **A Course in Phonetics**. 4th ed. Harcourt Brace, 2001.

LANDER, T; COLE, R. A; OSHIKA, B. T; NOEL, M. The OGI-22 Language Telephone Speech Corpus. In: EUROPEAN CONFERENCE ON SPEECH COMMUNICATION AND TECHNOLOGY, 4th, 1995, Madrid. **Proceedings…** Madrid: [s.n], 1995.

LEE, C; SINISCALCHI, S. M. An Information-Extraction Approach to Speech Processing: Analysis, Detection, Verification, and Recognition. **Proceedings of the IEEE**, v.101, n.5, p.1089-1115, 2013.

LEE, C. H; GAUVAIN, J. L. Speaker Adaptation Based on MAP Estimation of HMM Parameters. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 1993, Minneapolis. **Proceedings…** Minneapolis: IEEE, 1993. p.558-561.

LEE, D; JEONG, M; KIM, K; RYU, S; LEE, G. G. Unsupervised Spoken Language Understanding for a Multi-Domain Dialog System. **IEEE Transactions on Audio, Speech, and Language Processing**, v.21, n.11, p.2451-2464, 2013.

LESTER, J; BRANTING, K; MOTT, B. Conversational Agents. In: SINGH, M. P. (Ed.). **The Practical Handbook of Internet Computing**. 2004. p.220-240.

LITMAN, D. J; PAN, S. Designing and Evaluating an Adaptive Spoken Dialogue System. **User Modeling and User-Adapted Interaction**, v.12, n.2, p.111-137, 2002.

LOPES, J; ESKENAZI, M; TRANCOSO, I. Automated Two-Way Entrainment to Improve Spoken Dialog System Performance. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 2013, Vancouver. **Proceedings…** Vancouver: IEEE, 2013. p.8372-8376.

LÓPEZ-CÓZAR, R; SILOVSKY, J; KROUL, M. Enhancement of Emotion Detection in Spoken Dialogue Systems by Combining Several Information Sources. **Speech Communication**, v.53, n.9, p.1210-1228, 2011.

MORGAN, J; ACKERLIND, S; PACKER, S. **West Point Brazilian Portuguese Speech**. Linguistic Data Consortium, Philadelphia, 2008. Disponível em: <http://catalog.ldc.upenn.edu/LDC2008S04>. Acesso em: 07 dez. 2013.

NETO, N; SILVA, P; KLAUTAU, A; ADAMI, A. Spoltech and OGI-22 Baseline Systems for Speech Recognition in Brazilian Portuguese. **Lecture Notes in Computer Science**, v.5190, p.256-259, 2008a.

NETO, N; SIRAVENHA, C; MACEDO, V; KLAUTAU, A. A Computer-Assisted Learning Software to Help Teaching English to Brazilians. In: INTERNATIONAL CONFERENCE ON COMPUTATIONAL PROCESSING OF THE PORTUGUESE LANGUAGE, Special Session, 2008, Curia. **Proceedings…** Curia: [s.n.], 2008b.

OLIVEIRA, R; BATISTA, P; NETO, N; KLAUTAU, A. Recursos para Desenvolvimento de Aplicativos com Suporte a Reconhecimento de Voz para Desktop e Sistemas Embarcados. In: WORKSHOP DE SOFTWARE LIVRE, 2011, Porto Alegre. **Proceedings…** Porto Alegre: [s.n.], 2011.

_____. Baseline Acoustic Models for Brazilian Portuguese Using CMU Sphinx Tools. **Lecture Notes in Computer Science**, v.7243, p.375-380, 2012.

QADEER, M. A; IMRAN, A. Asterisk Voice Exchange: An Alternative to Conventional EPBX. In: INTERNATIONAL CONFERENCE ON COMPUTER AND ELECTRICAL ENGINEERING, 2008, Phuket. **Proceedings…** Phuket: IEEE, 2008. p.652-656.

QIN, D. Research on the Performance of Asterisk-Based Media Gateway. In: INTERNATIONAL SYMPOSIUM ON KNOWLEDGE ACQUISITION AND MODELING, 4th, 2011, Sanya. **Proceedings…** Sanya: IEEE, 2011. p.347-349.

RABINER, L; JUANG, B. **Fundamentals of Speech Recognition**. Prentice Hall, 1993.

SADEK, D. Design Considerations on Dialogue Systems: From Theory to Technology – The Case of Artimis. In: WORKSHOP ON INTERACTIVE DIALOGUE IN MULTI-MODAL SYSTEMS, 1999, Kloster Irsee. **Proceedings…** Kloster Irsee: ISCA, 1999. p.173-187.

SCHEFFLER, K. H. **Automatic Design of Spoken Dialogue Systems**. 2002. 117p. Dissertation (Doctor of Philosophy Degree) – University of Cambridge, Cambridge, 2002.

SCHRAMM, M; FREITAS, L; ZANUZ, A; BARONE, D. **CSLU: Spoltech Brazilian Portuguese Version 1.0**. Linguistic Data Consortium, Philadelphia, 2006. Disponível em: <http://catalog.ldc.upenn.edu/LDC2006S16>. Acesso em: 07 dez. 2013.

SILVA, D. C; LIMA, A. A; MAIA, R; BRAGA, D; MORAES, J. F; MORAES, J. A; RESENDE, F. G. V. A Rule-Based Grapheme-Phone Converter and Stress Determination for Brazilian Portuguese Natural Language Processing. In: INTERNATIONAL TELECOMMUNICATION SYMPOSIUM, 2006, Fortaleza. **Proceedings...** Fortaleza: IEEE, 2006. p.550-554.

SILVA, P; NETO, N; KLAUTAU, A; ADAMI, A; TRANCOSO, I. Speech Recognition for Brazilian Portuguese Using the Spoltech and OGI-22 Corpora. In: SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 2008, Rio de Janeiro. **Anais...** Rio de Janeiro: [s.n.], 2008.

SILVA, P; NETO, N; KLAUTAU, A. Novos Recursos e Utilização de Adaptação de Locutor no Desenvolvimento de um Sistema de Reconhecimento de Voz para o Português Brasileiro. In: SIMPÓSIO BRASILEIRO DE TELECOMUNICAÇÕES, 2009, Blumenau. **Anais...** Blumenau: [s.n.], 2009.

SILVA, P; BATISTA, P; NETO, N; KLAUTAU, A. An Open-Source Speech Recognizer for Brazilian Portuguese with a Windows Programming Interface. **Lecture Notes in Computer Science**, v.6001, p.128-131, 2010.

SIRAVENHA, A; NETO, N; MACEDO, V; KLAUTAU, A. Uso de Regras Fonológicas com Determinação de Vogal Tônica para Conversão Grafema-Fone em Português Brasileiro. In: INTERNATIONAL INFORMATION AND TELECOMMUNICATION TECHNOLOGIES SYMPOSIUM, 7th, 2008, Foz do Iguaçu. **Proceedings…** Foz do Iguaçu: [s.n.], 2008.

SIRAVENHA, A; NETO, N; MACEDO, V; KLAUTAU, A. A Computer-Assisted Learning Software Using Speech Synthesis and Recognition in Brazilian Portuguese. In: INTERACTIVE COMPUTER AIDED BLENDED LEARNING INTERNATIONAL CONFERENCE, 2009, Florianópolis. **Proceedings…** Florianópolis: [s.n.], 2009.

SONNTAG, D; HUBER, M; MÖLLER, M; NDIAYE, A; ZILLNER, S; CAVALLARO, A. Design and Implementation of a Semantic Dialogue System for Radiologists. In: HAFFNER, K. A. (Ed.). **Semantic Web: Standards, Tools and Ontologies**. 2010. p.41-66.

SUBRAMANIAM, L. V. Call Centres of the Future. **IT Magazine**, p.48-51, February 2008.

TAYLOR, P. **Text-to-Speech Synthesis.** Cambridge University Press, 2009.

TOKUDA, K; NANKAKU, Y; TODA, T; ZEN, H. Speech Synthesis Based on Hidden Markov Models. **Proceedings of the IEEE**, v.101, n.5, p.1234-1252, 2013.

TRAUM, D. Approaches to Dialogue Systems and Dialogue Management. **Notas de Aulas.** 2008. Disponíveis em: <http://people.ict.usc.edu/~traum/ESSLLI08>. Acesso em: 08 dez. 2013.

_____. Approaches to Dialogue Systems and Dialogue Management. **Palestra.** 2012. Disponível em: <http://people.ict.usc.edu/~traum/Talks/cs544dialogue3-8-12.pdf>. Acesso em: 08 dez. 2013.

TSAI, M. The VoiceXML Dialog System for the E-Commerce Ordering Service. In: INTERNATIONAL CONFERENCE ON COMPUTER SUPPORTED COOPERATIVE WORK IN DESIGN, 9th, 2005, Coventry. **Proceedings…** Coventry: IEEE, 2005. p.95-100.

WANG, F; SWEGLES, K. Modeling User Behavior Online for Disambiguating User Input in a Spoken Dialogue System. **Speech Communication**, v.55, n.1, p.84-98, 2013.

WEIZENBAUM, J. ELIZA – A Computer Program for the Study of Natural Language Communication between Man and Machine. **Communication of the ACM**, v.9, n.1, p.36-45, 1966.

YOSHINO, K; MORI, S; KAWAHARA, T. Incorporating Semantic Information to Selection of Web Texts for Language Model of Spoken Dialogue System. In: IEEE INTERNATIONAL CONFERENCE ON ACOUSTICS, SPEECH AND SIGNAL PROCESSING, 2013, Vancouver. **Proceedings…** Vancouver: IEEE, 2013. p.8252-8256.

YOUNG, S; EVERMANN, G; GALES, M; HAIN, T; KERSHAW, D; LIU, X. A; MOORE, G; ODELL, J; OLLASON, D; POVEY, D; VALTCHEV, V; WOODLAND, P. **The HTK Book (for HTK Version 3.4).** Cambridge University Engineering Department, 2006.

YOUNG, S; GASIC, M; THOMSON, B; WILLIAMS, J. D. POMDP-Based Statistical Spoken Dialog Systems: A Review. **Proceedings of the IEEE**, v.101, n.5, p.1160-1179, 2013.

ZUE, V. W; GLASS, J. R. Conversational Interfaces: Advances and Challenges. **Proceedings of the IEEE**, v.88, n.8, p.1166-1180, 2000.

ZUE, V. Eighty Challenges Facing Speech Input / Output Technologies. In: FROM SOUND TO SENSE: 50+ YEARS OF DISCOVERIES IN SPEECH COMMUNICATION CONFERENCE, 2004, Cambridge. **Proceedings…** Cambridge: MIT, 2004.

GLOSSÁRIO

Audiobook

Um *audiobook* é uma gravação, em áudio, do conteúdo de um livro lido em voz alta por um locutor.

Baseline

Baselines são sistemas de referência que servirão para avaliação de novos sistemas através de comparação. Por exemplo, se o sistema *baseline* possui uma taxa de erro por palavra de 30%, um novo sistema que obtenha 32% de taxa de erro por palavra não deve ser continuado, pois é pior que o sistema tomado como referência.

Cepstral

Cepstral é relativo a *cepstrum*, que foi obtida invertendo-se a primeira metade da palavra “*spectrum*”, que significa “espectro”. O *cepstrum* plota a amplitude de um sinal em função do inverso de sua frequência.

Corpora

Plural de *corpus.*

Corpus

Conjunto de documentos que servem de base para a descrição ou o estudo de um fenômeno. Tais documentos podem ser texto ou áudio, por exemplo.

Desktop

Desktop é o computador de mesa, composto basicamente pelo gabinete, monitor, teclado e *mouse*. Aplicações *desktop* são programas de computador que não dependem de um navegador de Internet e rodam de maneira independente na máquina do usuário.

Engine

Uma *engine* é um *software* pronto que contém código relativo a funções essenciais para aplicativos de um determinado tipo. As *engines* servem como núcleo

para o desenvolvimento de outras aplicações maiores. Uma *engine* para reconhecimento de voz lida com todo o processamento para reconhecer, de fato, o áudio gerado pelo locutor. Ela pode ser utilizada para o desenvolvimento de diversas aplicações diferentes como, por exemplo, um navegador de Internet comandado através da fala.

Framework

Framework é um conjunto de códigos que colaboram entre si para executar uma responsabilidade que resolva um problema específico dentro do desenvolvimento de aplicações. O *framework* dita o fluxo de controle da aplicação que o utiliza. Pode ser entendido como um molde para aplicações de um determinado tipo, como, por exemplo, aplicações de voz sobre IP (VoIP).

Hardware

Parte física de um computador, formada pelos componentes eletrônicos e mecânicos necessários para seu bom funcionamento.

Help Desk

Serviço de apoio a usuários para suporte e resolução de problemas técnicos em um domínio específico, seja informática, telefonia, vendas, entre outros. Pode ser realizado presencialmente, por telefone, por e-mail, ou por qualquer meio de comunicação.

Java

Linguagem de programação, amplamente utilizada, orientada a objetos e de código aberto.

Know-how

Conjunto de conhecimentos práticos adquiridos por uma empresa ou profissional, que traz pra si vantagens competitivas.

Laptop

Laptop é um computador portátil, também conhecido por *notebook*. Ele possui as mesmas funcionalidades de um *desktop* reunidas em um único dispositivo.

Mouse

Periférico de entrada de dados muito comumente utilizado em computadores. Serve para movimentar um cursor através da tela e interagir através de cliques.

Plugin

Um *plugin* é um módulo de extensão, isto é, um programa de computador usado para adicionar funções a outros programas maiores, provendo funcionalidades específicas.

Prompt

Arquivo de áudio contendo uma gravação a ser executada.

Proxy

Proxy é um servidor intermediário que repassa as requisições que recebe para outro servidor. O *proxy* pode ter diversas funcionalidades adicionais como tratamentos de segurança relacionados às requisições que passam por ele.

Script

Scripts são como roteiros a serem executados por uma aplicação. O programa principal se utiliza dos *scripts* para executar ações em determinado momento. Por exemplo, o Asterisk utiliza o *script dialplan* para controlar, de acordo com o desejo do usuário, uma ligação.

Smartphone

São telefones celulares providos de sistema operacional que permite se adicionar ao aparelho inúmeras novas funcionalidades através de aplicativos externos. Além disso, os *smartphones* podem se conectar à Internet.

Smart TV

São aparelhos de televisão que podem ser conectar à Internet para prover novas funcionalidades aos usuários.

Software

Software é uma sequência de instruções escritas para serem interpretadas por um computador com o objetivo de executar tarefas específicas. Em outras palavras, *softwares* são os programas de computadores.

Tablet

Dispositivo eletrônico pessoal em formato de prancheta que pode ser usado para diversos fins como acesso à Internet, leitura de livros e jogos, por exemplo.

Tag

Tags são marcações que trazem significado semântico para um determinado texto constante em um documento escrito em alguma linguagem de marcação. Elas funcionam como etiquetas que identificam que tipo de informação elas contêm.

Touch Screen

Telas de dispositivos eletrônicos sensíveis ao toque e que podem ser comandadas dessa forma.

Voice Browser

Navegador de voz. São análogos aos navegadores de Internet, porém são capazes de interpretar documentos VoiceXML, em vez de documentos HTML.

Wizard

Wizard é um programa de computador cuja função é servir de assistente ao usuário para que este consiga, de maneira simples, realizar tarefas complexas. O *wizard* guia o usuário através do passo a passo das tarefas a serem executadas. Este tipo de assistente é comumente encontrado em instaladores de programas.

World Wide Web

O termo World Wide Web (WWW) se refere de maneira ampla a toda a rede de computadores interligados através da Internet. É comumente traduzido como rede mundial de computadores.