Proposta de Arquitetura Utilizando o Paradigma SOA para o Framework FIVE

Architecture proposal using the SOA paradigm for framework FIVE

Davino Wagner da Silva Mariano [1] orcid.org/0000-0002-7391-6701

Alexandre Magno Andrade Maciel [1] orcid.org/0000-0003-4348-9291

[1] Escola Politécnica de Pernambuco, Universidade de Pernambuco, Recife, Brasil

E-mail do autor principal: Davino Wagner da Silva Mariano davinowagner@gmail.com

Resumo

O desenvolvimento de interface de voz tem recebido grande atenção do meio acadêmico em virtude da interação humano-computador e a busca por sistemas que se adequem como prótese aos usuários auxiliando em suas atividades quer sejam para reconhecimento ou síntese, além da necessidade de aplicações robustas para o trabalho de reconhecimento da fala e síntese de voz, é exequível outras áreas do conhecimento para compreensão análise e desenvolvimento. Este trabalho apresenta "Uma proposta de arquitetura utilizando o paradigma SOA (Service Oriented Architecture) para o framework FIVE (Framework for an Integrated Voice Environment)'', onde foi empregado os conceitos da arquitetura REST (Representational State Transfer) para validar um modelo de arquitetura na qual foi realizado testes para avaliar o comportamento das alterações propostas, uma solução de software para fornecer multi-plataforma de acesso através dos recursos web, usando reconhecimento e síntese de voz de forma integrada.

Palavras-Chave: *FIVE; SOA; Rest; Multi-Plataforma;*

Abstract

The development of voice interface has received great attention from the academic environment due to the human-computer interaction and the search for systems that fit as prosthesis to the users assisting in their activities, whether for recognition or synthesis, and the need for robust applications for The work of speech recognition and speech synthesis, is feasible other areas of knowledge for understanding analysis and development. This paper presents an architecture proposal using the Service Oriented Architecture (SOA) paradigm for the Framework for an Integrated Voice Environment (FIVE) framework, where the concepts of the REST (Representational State Transfer) architecture were used to validate a model of Architecture in which tests were performed to evaluate the behavior of proposed changes, a software solution to provide multi-platform access through web resources, using integrated speech recognition and synthesis.

Key-words: *FIVE; SOA; Rest; Multi-Platform;*

http://dx.doi.org/10.25286/repa.v2i2.558

1 Introdução

A fala é uma das mais importantes formas de interação entre os seres humanos visto que ela é essencial para a transmissão de informações e para a aquisição de conhecimento. Tornar os computadores capazes de interagir por meio de voz humana é um objetivo perseguido há muito tempo, e conseguir isto permanece um grande desafio até hoje **[1]**. A compreensão de sotaques diferentes e a produção de voz sintética com qualidade natural, além de alta performance no tratamento dos dados recebidos, são algumas das particularidades e necessidades especiais **[2]**.

A área de interface de voz com o usuário (no inglês: *Voice User Interface – VUI*) tem recebido atenção de diferentes grupos de pesquisa em virtude de dois principais fatores: primeiro, em virtude de melhorias no desempenho de sistemas de processamento automático e de reconhecimento de fala, tradução de idiomas falados e síntese de voz; segundo, devido a convergência de dispositivos e a sólida produção de conteúdos multimídia, que exigem uma interação otimizada com os usuários **[3]**.

Apesar dos inúmeros avanços obtidos nesta área, poucos esforços tem sido feitos a fim de trazer essas contribuições para o nível de aplicação **[4]**. Diante disto, foi desenvolvido o FIVE (*Framework for an Integrated Voice Environment*) com o objetivo de auxiliar o desenvolvimento integrado de aplicações com interface de voz em diferentes ambientes tecnológicos (*call centers*, dispositivos móvel, TV digital), com rápida curva de aprendizagem, independência de plataforma e extensível para novas técnicas **[5]**.

A arquitetura do FIVE é composta por três camadas: CORE, API e GUI. Na camada CORE estão disponíveis algoritmos avançados para extração de características, classificação de padrões de voz e para a geração de motores de fala. Já a camada API consiste numa implementação própria que disponibiliza os recursos necessários à instanciação dos motores de fala. Na camada GUI é disponibilizado uma interface gráfica em formato de *wizard* a fim de facilitar o processo de construção e teste dos motores gerados **[5]**.

Esta arquitetura, apesar de funcional, apresenta alguns entraves na oferta dos motores de fala, que são: baixa escalabilidade para novas técnicas; e dificuldade de manutenção no código fonte do projeto. A escolha de uma arquitetura orientada a serviços (no inglês: *Service Oriented Architecture - SOA*) proporcionaria a flexibilidade necessária para a criação de produtos com fraco acoplamento, facilitando a escalabilidade e a manutenibilidade do serviço **[6]**.

O presente trabalho tem como objetivo realizar a adaptação do framework FIVE para a arquitetura SOA, de modo a obter maior escalabilidade aos motores de fala gerados, proporcionando maior manutenabilidade no projeto e tornando-o aderente padrão de projeto MVC(Model View Control). Para isto será realizado um estudo pormenorizado da abordagem SOA e do Framework FIVE (Seção 2), em seguida realizada a refatoração da arquitetura do FIVE para torná-la orientada a serviço em uma aplicação web (Seção 3), Em seguida será realizada um processo de construção e instanciação de um motor de fala para validação da arquitetura proposta (Seção 4), e por fim, a Seção 5 apresenta as conclusões e trabalhos futuros.

2 Fundamentação Teórica

Cada parte do manuscrito vai ser comentada detalhadamente nos seguintes numerais. Leia e siga as instruções de maneira cuidadosa, a primeira revisão Editorial é uma revisão de forma, caso o manuscrito esteja fora das diretrizes o artigo será rejeitado automaticamente.

2.1 Arquitetura Orientada a Serviços (SOA)

A arquitetura orientada a serviços não é mais uma novidade na indústria de software, contudo, ela ainda representa uma abordagem da computação distribuída com grande aderência aos projetos de software devido ao fraco acoplamento de serviços, interfaces de publicáveis e governados, e modelo de comunicação padrão **[7]**.

Segundo OASIS1 **[8]**, SOA consiste num paradigma para organização e utilização de recursos distribuídos que podem estar sob o controle de diferentes domínios proprietários. Ele interage com a capacidade de produzir efeitos desejados consistentes com pré-condições e expectativas mensuráveis. Já para SILVA **[9]** SOA é definida como uma arquitetura que promove a integração do negócio com a tecnologia da informação, utilizando componentes de serviços, obtendo como resultados agilidade para atender as novas demandas, fornecendo flexibilidade nas mudanças, auxiliando na redução de custo, provendo reuso de componentes para software e serviço.

O uso da SOA levou a grandes avanços na integração de aplicações. Alguns exemplos usuais de serviços com características SOA são: consultas de nomes e endereços, abertura de conta no ambiente

bancário, agendamento de consultas médicas, consulta a multas de trânsito e etc. Contudo, existem aplicações utilizando este mesmo paradigma, que são muito mais robustas, que exploram oportunidades para propor o aumento da produtividade e a integração entre sistemas, com diversas tecnologias, por exemplo, o *Oracle Service Bus 12c* permite, por meio de um suíte SOA que as empresas forneçam para dispositivos móveis funcionalidades de suas plataformas de integração, ou seja, os desenvolvedores podem simplificar o processo de criação de aplicativos personalizáveis a partir de componentes reutilizáveis **[10]**.

Nos últimos anos uma dentre muitas abordagens para implementação da SOA que tem obtido grande aceitação no desenvolvimento de software é o REST (*Representational State Transfer*) **[11]**, em português Transferência de Estado Representacional. REST é um estilo arquitetural para o desenvolvimento de sistemas hipermídias distribuídos, proposto por Roy Thomas Fielding, um dos especificadores do protocolo HTTP 1.1. Este modelo de implementação não se trata de especificação ou padrão, mas sim um paradigma para o desenvolvimento e construção de sistemas distribuídos. Aplicadas a componentes, conectores e elementos de dados dentro de um sistema **[12]**.

No estilo arquitetônico REST, dados e funcionalidades são considerados recursos, e esses recursos são acessados através de URI (*Uniform Resource Identifier*), conhecidos tipicamente como links na web. Os recursos são representados em prática através de um conjunto simples, de operações bem definidas. A Figura 1 mostra uma visão geral da arquitetura REST em um modelo cliente servidor.

Arquitetura REST foi projetada para utilizar um protocolo de comunicação padronizado, do tipo HTTP. Este protocolo estimula aplicações *RESTful* a serem simples, leve e com alta performance. Esta arquitetura fornece mecanismos para abstrair detalhes de sua implementação exigindo esforços para ajustes em componentes conectores e dados. Aplicações em *RESTful* tipicamente utilizam os quatro principais métodos HTTP em suas operações são eles: GET, POST, PUT e DELETE. O Quadro 1, descreve o uso dos principais métodos utilizados pelo REST.

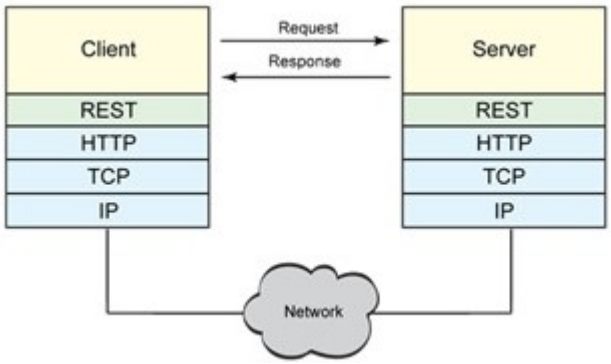

Figura 1: Visão geral da arquitetura REST.
Fonte: http://www.ibm.com/developerworks/br/library/os-understand-rest-ruby/

Quadro 1: Métodos utilizados pelo REST.

Método	Operação
GET	Buscar um recurso.
POST	Usado para criar recursos e outras operações, não definidas semanticamente.
PUT	Criar ou atualizar um recurso.
DELETE	Deletar/apagar um recurso.

Fonte: O autor.

2.2 Arquitetura Implementada no FIVE

A arquitetura do FIVE é composta por três camadas. A camada CORE é formada por um conjunto de classes abstratas, que servem de interface para a implementação de inúmeras técnicas para extração de características, tal como MFCC (*Mel-frequency Cepstral Coefficients*) umas das adotadas na aplicação, mas existem outras; e para classificação de padrões, tal como HMM (*Hidden Markov Models*), também muito utilizada. CORE foi desenvolvido com base na abordagem de framework gray-box (Uma abordagem de herança e composição composta geralmente por classes abstratas e concretas). A Figura 2 mostra o diagrama de classe do CORE.

A camada API figura 3, consiste na implementação de um conjunto de classes necessárias para à comunicação dos motores de fala gerados, e das aplicações que por ventura irão utilizá-los. Esta

http://dx.doi.org/10.25286/repa.v2i2.558

camada está dividida em dois pacotes, um de reconhecimento e outro de síntese.

A camada GUI do framework consiste numa aplicação com interface baseada em *Wizard*, responsável por todo o passo a passo para preenchimento do cadastro básico e validação dos campos com informações para a construção dos motores de fala. Foi implementado com padrão de projeto *facade*, que é um padrão com características do tipo estrutural. O uso desse padrão ajuda o desenvolvedor a compreender melhor a API já que é intrínseco ao padrão produzir interfaces simples para as funcionalidades do sistema, o que facilita o processo de desenvolvimento. A Figura 4 mostra os pacotes existentes na GUI e a classe *facade*.

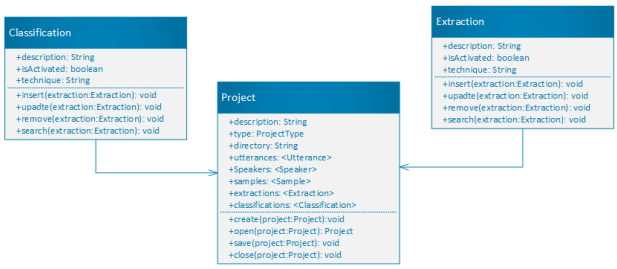

Figura 2: Diagrama de classes do CORE.
Fonte: O autor

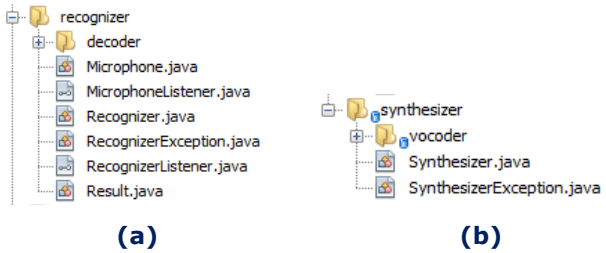

Figura 3: Diagrama de classes da API.
Fonte: O autor

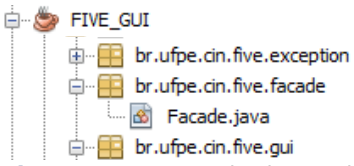

Figura 4: Conjunto de classes da API.
Fonte: O autor

O FIVE, apesar de atender os objetivos iniciais da proposta do trabalho em Maciel **[5]**, não oferecia subsídios para a escalabilidade e manutenibilidade do software, visto que a falta de separação de responsabilidades, na camada GUI, causa esse *gap.* Ou seja, caso uma nova técnica fosse adicionada era necessário realizar modificações em todas as camadas do sistema. Levando em consideração estes detalhes, foi escolhida a SOA como arquitetura, para prover solução e organização da API como serviço. Algumas atividades realizadas na antiga arquitetura tornava menos produtiva a execução das tarefas de geração e manipulação dos *engines*, já que os mesmos eram gerados e copiados de forma manual e posteriormente salvos no servidor de aplicação sem integração e controle durante o fluxo da operação.

3 Arquitetura Proposta

A arquitetura proposta implementa uma camada de web service (FIVE WS) com os conceitos da arquitetura SOA, bem como a extração das regras de negócios da camada FIVE GUI para uma nova camada FIVE CONTROLLER. Onde é esperado manter separada as responsabilidades entre as camadas de regras de negócios e a camada GUI. A Figura 5 mostra o desenho da nova arquitetura desenhada para exemplificar a estrutura e modularização do projeto. A camada *controller* intermédia nas requisições de request e response, é também responsabilidade do controller cuidar dos *models* e *views* para supervisionar as solicitações de apresentação dos dados. Com isso vamos ter uma camada mais leve, com código coeso e baixo acoplamento. Essa separação de responsabilidades favorece uma construção da arquitetura aderente a componentização de micro-serviços. Tornando o software um produto com baixo custo para manutenção.

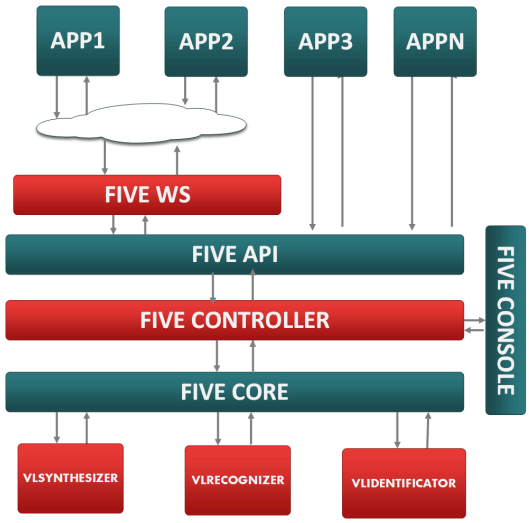

Figura 5: Visão geral da arquitetura REST.
Fonte: O autor

A criação da camada FIVE WS propõe tornar o FIVE mais aderente ao processo de escalabilidade para criação de novos serviços e contratos de interfaces. Os outros módulos não serão impactados, pois serão isolados pela FIVE API. Já a criação da camada FIVE CONTROLLER, irá proporcionar uma melhor manutenibilidade inserindo neste contexto a separação de conceitos, onde será realizada todo controle de fluxo na ações de exibição de telas, e chamadas partes de modelos do sistema, está camada terá o papel de intermediador entre as requisições.

3.1 Tecnologias Utilizadas

As tecnologias utilizadas neste projeto foram: Java (versão 8) para linguagem de programação e construção do backend, IDE Netbeans para ambiente de desenvolvimento, servidor web Glassfish para publicação da aplicação, Postman para realização de testes unitários e consultas na API REST, Loader.io e Apache JMeter para testes de carga e performance.

3.2 Implementação do FIVE WS

O FIVE WS foi construído em módulos, o sistema permanece separados em pacotes. O *config* corresponde ao módulo de configuração da aplicação onde é parametrizada a classe para configuração do servidor, o *controller*, implementa os métodos das interfaces criadas no pacote *service.* Este modelo de implementação obriga o desenvolvedor a primeiro criar as interfaces e não implementações, afim de manter a organização e limpeza do código. A Figura 6 mostra a visão do projeto FIVE WS (a), bem como as definições dos cabeçalhos do serviço na Figura (b).

Figura 6-a: FIVE_WS: Classes do controller

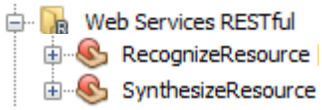

Figura 6-b: FIVE_WS: Recursos do RESTful.

3.3 Implementação do FIVE CONTROLLER

A camada *controller* no FIVE foi desenvolvida com o objetivo de atuar como um intermediador das requisições entre as *views* e regras de negócios da aplicação existentes nos models. O controlador resolver alguns problemas por meio da separação de tarefas, tanto no acesso aos dados, como na lógica de negócio, e na camada de presentação que realiza interação com o usuário. Na Figura 7 temos a visão do módulo controller.

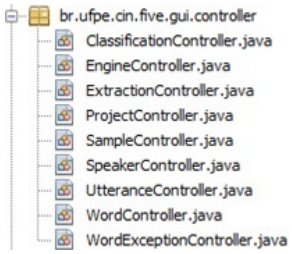

Figura 7: Pacote controller.
Fonte: O autor.

No pacote controller são disponibilizadas as classes: *ClassificationController* que realiza a integração entre as classes Sample e Extraction, nessa mesma classe é realizada a classificação das *samples* (amostras de áudio) extraídos no método processSpeechClassification. A classe *EngineController* que verifica a existência dos *engines* e suas validações por exemplo: Id dos *engines*, ocorrências na extração de características e etc. Já na classe *ExtranctionController* ocorre o acompanhamento das extrações de características por iteração de uma lista onde são verificados os tipos de técnicas aplicadas nas amostras coletadas. A classe *ProjectController* desempenha a função de

http://dx.doi.org/10.25286/repa.v2i2.558

criação e gerência dos projetos no *framework.* Com o *SampleController* é possível realizar busca nas amostras de áudio, como também tratativas das *features* do projeto. Dentro da *SpeakerController* é feita validação dos ***locutores***, busca e manutenção com os serviços de insert, update, delete e retrieve. Já a classe *UtteranceController* realiza a tratativa nas frases, sílabas e outras palavras, que também são feitas as operações de insert, delete e update dos conjuntos de palavras reconhecidas.

4 Validação da Arquitetura

4.1 Camada Controller

Para validação da camada Controller foi realizado o processo de construção de um motor de reconhecimento de fala isolada. Foram selecionadas quatro locuções com suas respectivas amostras de áudio (abrir, fechar, subir e descer). O processo de extração de características e de classificação de padrões foi realizado por meio das técnicas de MFCC (*Mel-Frequancy Cepstral Coeficientes*) e HMM (*Hidden Markov Models)*. Ao final do processo de treinamento e teste obteve-se uma taxa de acerto de 92,5%, aceitável para os objetivos deste trabalho que é apenas da validação da arquitetura. Ao final, foi gerado o motor de reconhecimento e utilizada uma aplicação de teste para validar o motor gerado. A Figura 8 mostra a estrutura de diretórios contendo os arquivos do motor gerado.

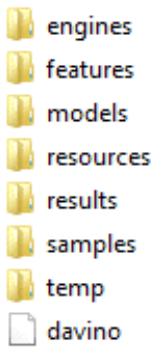

Figura 8: Estrutura de diretórios do motor.
Fonte: O autor.

4.2 Camada WS

O processo de validação da camada WS foi realizado por meio da criação de um protótipo para realização de testes no consumo do serviço web criado. O motor usado nessa fase foi o VL Synthesizer construído em Souza *et al.* **[15]**. A Figura 9 mostra a chamada ao serviço mediante a url apresentada no browser com o endereço do recurso. Neste momento é passada a amostra de áudio e o motor gerado como parâmetros na requisição REST, posteriormente o servidor retorna uma amostra de wave bem como o resultado da síntese de voz obtido neste caso a palavra (ABRIR).

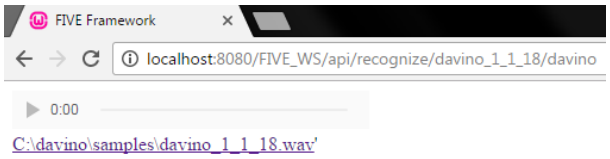

Figura 9: Recognize Abrir.
Fonte: O autor.

Na figura 10 é apresentada a chamada do serviço *recognize* onde é executado o reconhecimento da amostra de voz fechar coletada e armazenada no diretório *samples*. Por meio da execução do serviço implementado na API.

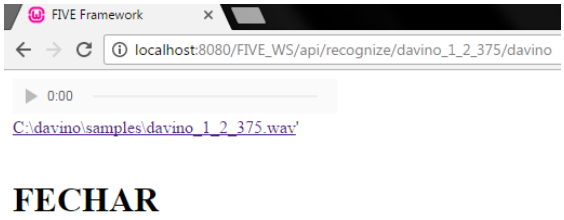

Figura 10: Recognize Fechar.
Fonte: O autor.

Os parâmetros passados são: amostra de áudio e o *engine* desejado para execução do reconhecimento.

4.3 Testes

No plano de teste proposto para aferir o tempo de resposta nas requisições enviadas para o endpoint, foi executado utilizado o POSTMAN a Figura 11 apresenta uma sessão de testes efetuada na ferramenta. A média obtida no tempo de reposta foi de 2712 milisegundos após realização de 500 requisições a API se comportou de maneira estável, mas quando submetido a uma carga de 1000 requisições o servidor travou.

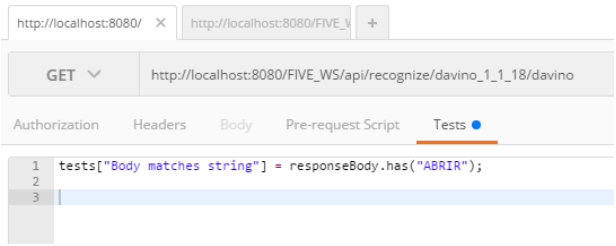

Figura 11: Postman teste unitário.
Fonte: O autor.

4.4 Discussão

Os experimentos realizados validaram as alterações sugeridas na proposta deste trabalho. Para este fim, os resultados obtidos foram considerados satisfatórios visto que nenhum erro foi encontrado e as taxas de acerto dos motores permaneceram as mesmas quando apresentadas em outros trabalhos. Desse modo, consideramos que as melhorias proposta atingiram os propósitos da pesquisa. Apesar dos bons resultados o estudo limita-se apenas em disponibilizar recursos já existentes na API em uma nova plataforma web sendo assim os conceitos utilizados na construção do FIVE foram apenas ajustados dentro do contexto para serviços. A pretensão de execução deste trabalho é de proporcionar de forma sistêmica, clara e prática como dar-se-á o uso da nova arquitetura aplicada ao framework com a finalidade de facilitar a produção de *engines* e execução dos recursos no ambiente web. Uma vez que intensificando a sua utilização irá fomentar o uso do framework e a sua demanda na área de pesquisa sobre interface de voz no meia acadêmico.

5 Conclusões

O experimento realizado neste trabalho obteve êxito, tendo em vista que as alterações proposta para o framework funcionaram de acordo com os resultados comparados em outros trabalhos como Maciel **[5]**, Souza **[15]**, nos quais foram submetidos com o mesmo framework. Foi detectado após análise que dentro da camada *gui* do FIVE existe muito código relacionado ao contexto lógico. Para tanto foi realizado uma inspeção afim de projetar uma arquitetura que oferecesse a possibilidade de torna-lo escalável, mantendo organização no código, otimizando o processo de adição para novas *features*, e consequentemente à aplicando da separação de conceitos dentro do processo de requisição entre a camada *gui* e a *model*. Com isso foi possível obter ganhos relativos a produtividade, além de deixar a aplicação mais robusta, no que diz respeito ao cenário anterior. Está separação horizontal aplicada com a nova arquitetura, permitiu que a nova camada *controller*, realize as requisições de *request* e response, direcionado para a camada *gui* as *features* intrínsecas da *api*. Contudo o fato de ter separado a *gui* das regras de negócios vai possibilitar em trabalhos futuros a criação de um barramento de serviços, para consumo por meio de uma interface web onde será disponibilizado um wizard para realização de cadastros básicos, configurações dos parâmetros e criação de novos *engines*. Este modelo de arquitetura é baseada na MVC padrão que mantem a camada de visualização desacoplada das camadas, *model* e *controller*. Em trabalho futuros será possível realizar uma versão totalmente web do FIVE para realizar acessos via API REST ao *endpoint* FIVE WS.

Referências

[1] R. Oliveira, P. Batista, N. Neto, A. Klautau, Recursos para desenvolvimento de aplicativos com suporte a reconhecimento de voz para desktop e sistemas embarcados. XII Forum Internacional de Software Livre, 2011.

[2] YNOGUTI, Carlos Alberto. Reconhecimento de fala contínua usando modelos ocultos de Markov. 1999. Tese de Doutorado. Universidade Estadual de Campinas.

[3] SALVADOR, Valeria Farinazzo Martins; DE OLIVEIRA NETO, Joao Soares; KAWAMOTO, Andre Satoshi. Requirement engineering contributions to voice user interface. In: Advances in Computer-Human Interaction, 2008 First International Conference on. IEEE, 2008. p. 309-314.

[4] MACIEL, Alexandre; CARVALHO, Edson. FIVE–Framework for an Integrated Voice Environment. In: Proceedings of International Conference on Systems, Signal and Image Processing, Rio de Janeiro. 2010.

[5] MACIEL, A. M. A. Investigação de um ambiente para o desenvolvimento integrado de interface de voz. 2012. Tese de Doutorado. Tese (Doutorado em Ciência da Computação). Universidade Federal de Pernambuco.

http://dx.doi.org/10.25286/repa.v2i2.558

[6] Martins, V. F., dos Santos, A. G., Rodrigues, F. A., Okumura, M. H., Sakoda, T. J., & de Paiva Guimarães, M. (2013). Análise, projeto e implementação de uma aplicação utilizando interface de voz com o usuário. Anais do Computer on the Beach, 41-50.

[7] AL SHEIKH, Maher A.; ABOALSAMH, Hatim A.; ALBARRAK, Ahmed. Migration of legacy applications and services to service-oriented architecture (SOA). In: The 2011 International Conference and Workshop on Current Trends in Information Technology (CTIT 11). IEEE, 2011. p. 137-142.

[8] SILVA, EDILBERTO. SOA - Arquitetura Orientada a Serviços: Conceitos e Aplicações. Acesso em 23/08/2016 URL http://www.edilms.eti.br/uploads/file/infrasft/unid05-is-soa.pdf.

[9] ARRABAL, Alejandro Knaesel. Publicação de artigos científicos. Prática da Pesquisa, set. 2010. Disponível em: <http://www.oracle.com/technetwork/middleware/service-bus/documentation/index.html>. Acesso em: 9 out. 2016.

[10] RICHARDSON, Leonard; RUBY, Sam. RESTful web services. " O'Reilly Media, Inc.", 2008.

[11] RODRIGUES, Lucas; DO PRADO, Antonio Francisco. Desenvolvimento de Aplicações Móveis com Serviços RESTful e HTML5. Revista TIS, v. 3, n. 2, 2014.

[12] ERL, Thomas. Soa: principles of service design. Upper Saddle River: Prentice Hall, 2008.

[13] FURTADO, Camille et al. Arquitetura Orientada a Serviço-Conceituação. Relatórios Técnicos do DIA/UNIRIO (RelaTe-DIA), RT-0012/2009, 2009.

[14] SOUZA, Danilo; SATURNINO, Levi; MACIEL, Alexandre MA. A Portability Evaluation of Brazilian Portuguese voices produced with MARY TTS. In: IWSSIP 2014 Proceedings. IEEE, 2014. p. 95-98.