Concepção e Implementação de um Gestor de Diálogos para um Tutor Artificial

Diana Costa (A78985), Gil Cunha (A77248) e Luís Costa (A74819)

Universidade do Minho - Campus de Gualtar - Departamento de Informática
Mestrado Integrado em Engenharia Informática
{a78985,a77249,a74819}@alunos.uminho.pt
http://www.di.uminho.pt/

21 de Junho de 2019

Resumo A ideia de automatizar e dimensionar diálogos de um para um atrai diversas marcas e tecnologias, que tentam proporcionar experiências ao utilizador simples, eficientes e adaptadas. No entanto, é frequentemente observável que os típicos *chatbots* acabam por "matar"o serviço ao cliente, tornando-se algo "detestável" e com baixos níveis de precisão. Na área da educação, um *chatbot* pode interagir com um aluno e ajudá-lo a estudar ou na realização de exercícios, fazendo com que aprender não seja tão monótono. O objetivo deste projeto foi integrar no projeto *Leonardo* uma personalidade cativante, e fazer com que este sistema fosse capaz de interatuar com o estudante universitário durante o seu estudo, com auxílio a características como ironia, incentivo ou informações sobre o desempenho do aluno. Depois do desenvolvimento desta componente autónoma, e através da geração automática e dinâmica de frases ao longo de todo o diálogo, a componente é capaz de improvisar e de se adaptar ao contexto. Porém, esta capacidade provém de um trabalho inicial manual e exaustivo, e ainda há espaço para melhoria da comunicação.

Keywords: Leonardo, Gestor de Diálogos, Avaliação, Padrão, Regras

1 Introdução

1.1 Contextualização

O projeto realizado consiste na implementação e conceção de um gestor de diálogos para um tutor artificial.

O tutor artificial, *Leonardo*, surge como um sistema de avaliação de determinadas unidades curriculares do ensino superior. O utilizador inicia o diálogo com o sistema através de um *chat* na plataforma existente e, para isso, tem de indicar o domínio e subdomínio que pretende abordar (um exemplo seria escolher o domínio de Base de Dados, com o subdomínio Álgebra Relacional). O sistema, após obter estas informações, começa a realizar perguntas e a interagir de acordo com as respostas obtidas.

Cada aluno que utilize o sistema *Leonardo* possui um perfil que indica várias informações sobre a sua avaliação nos vários temas disponibilizados, como o seu nível de desempenho e destreza.

– O **desempenho** representa a relação entre o número de respostas corretas com o número total de respostas, dadas pelo utilizador.

– A **destreza** representa a média do tempo global que o utilizador demora a responder a essas questões

A existência de um perfil por utilizador permite personalizar o diálogo especificamente para cada aluno e assim tornar a relação utilizador-sistema mais interessante e cativante aos olhos de quem o utiliza.

1.2 Motivação e Objetivos

A inexistência de um gestor de diálogos, responsável pela interação do sistema *Leonardo* com o utilizador, fazia com que a comunicação entre ambos se tornasse repetitiva e pouco apelativa ao aluno. Assim, surgiu a necessidade de tornar o sistema mais interativo.

Foi, então, decidido criar um gestor de diálogos capaz de personalizar a interação do sistema com o utilizador, incentivando, assim, à sua utilização através de diálogos mais dinâmicos e expeditos.

Consequentemente, este projeto tem como principal finalidade a criação de um componente autónomo capaz de trabalhar com outros módulos já existentes no sistema *Leonardo*, de uma forma natural e coesa. Para além disso, o componente tem de ser capaz de gerar frases capazes de suportar os vários processos do sistema, em termos gerais e em termos de cada domínio de estudo em particular, adequando-se à medida do contexto.

2 Trabalho Relacionado

Os seguintes projetos apresentam semelhanças ao realizado, visto que também têm como objetivo melhorar a interação entre um utilizador e o sistema em que este está inserido.

ChatBot [1] é um *bot* orientado para vendas que utiliza imagens e *links* como resposta a algumas perguntas dos seus utilizadores. Isto é algo inovador, tendo em conta que a sua interação não se limita a texto, o que torna o diálogo mais estimulante. O *ChatBot* utiliza também técnicas de *Machine Learning* que recorrem a algoritmos de Processamento de Linguagem Natural para analisar o *input* do utilizador e determinar semelhanças nas interações com os clientes.

Figura 1. ChatBot - "Automate customer service with ChatBot"

Duolingo [2] é um sistema de aprendizagem de idiomas que possui vários tutores, cada um com uma personalidade diferente, o que faz com que estes reajam de forma distinta às mesmas situações. Conforme o progresso do utilizador, este avança ao longo de diferentes níveis de habilidade que levam o estudante até ao fim do curso, enquanto oferece constantemente a opção de voltar atrás para repetir o estudo. A estrutura autodidata e interativa do método, semelhante a um jogo, foi a característica que mais cativou os utilizadores.

Figura 2. Duolingo - "Aprenda idiomas de graça"

3 Conceção e Implementação do Sistema

A componente desenvolvida consiste, portanto, num elemento individual e autónomo para posterior integração no sistema geral do projeto *Leonardo*. O seu papel será gerir o diálogo entre o *Leonardo* e os seus utilizadores, através de um *chat* na plataforma do projeto, de forma a conferir uma personalidade ao sistema.

A **arquitetura geral** do *Leonardo* é apresentada na imagem seguinte, onde se expõem cada uma das suas componentes e as relações entre estas, de um modo simplificado:

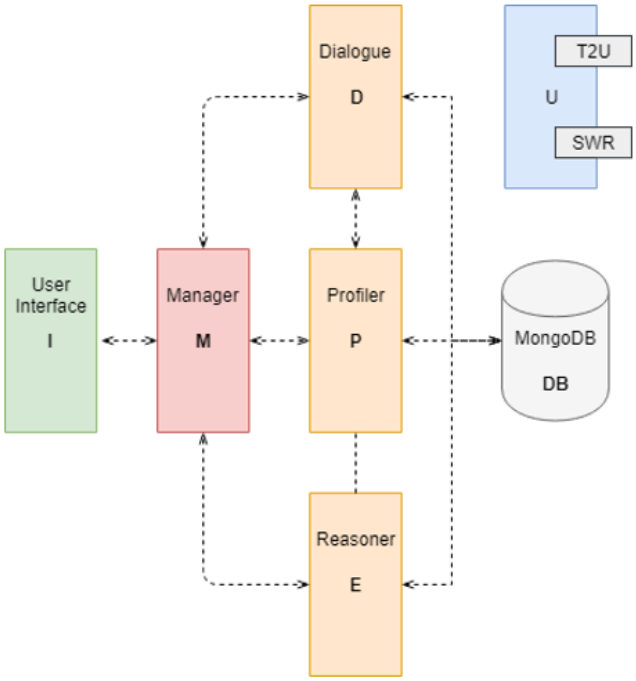

Figura 3. Arquitetura geral do sistema *Leonardo*

Para uma boa compreensão do funcionamento do sistema de gestão de diálogos elaborado, é importante perceber o conceito das seguintes componentes:

- ***Profiler***: componente que gere toda a informação dos utilizadores do sistema, como os seus perfis e avaliações;

- ***Reasoner***: componente responsável por calcular todas as métricas de avaliação de um utilizador quando este realiza um teste, como o seu desempenho e destreza;

- ***Manager***: tem a função de gerir tanto a comunicação entre as componentes do sistema como entre o sistema e os utilizadores. É, portanto, o responsável por controlar o funcionamento do *Leonardo*.

- ***Dialogue***: gere o diálogo entre o sistema e cada um dos utilizadores, durante um treino de avaliação.

É ainda de mencionar que todos as componentes referidas têm acesso a uma base de dados, de forma a armazenar e obter todos os dados necessários para o funcionamento do sistema geral.

3.1 Componente *Dialogue*

Salienta-se, assim, a componente ***Dialogue***, representante do gestor de diálogos desenvolvido, e a sua interação com a componente *Manager*. De uma forma geral, a função da componente *Dialogue* consiste na receção e processamento de uma **mensagem-padrão**, em formato *JSON*, que contem informação sobre o utilizador e a sua avaliação, em relação ao teste que e a sua interação com a componente *Manager*. De uma forma geral, a função da componente *Dialogue* consiste na receção e processamento de uma **mensagem-padrão**, em formato *JSON*, que contem informação sobre o utilizador e a sua avaliação, em relação ao teste que este está a realizar no momento. É a partir dos valores extraídos do padrão recebido, que o gestor de diálogos irá produzir e selecionar o diálogo mais adequado ao contexto. De referir que estes valores são proporcionados pela componente *Profiler*.

Assim, este processo pode ser dividido em 3 etapas:

Figura 4. As **3 etapas** do funcionamento do gestor de diálogos

- **Análise do padrão**: Esta primeira etapa consiste na receção da mensagem-padrão, enviada pelo *Manager*, e na análise dos seus dados. Esta análise, por sua vez, consiste na extração e conversão dos valores enviados na mensagem-padrão em informação útil para posterior processamento e criação de diálogo. Estes valores são, por exemplo, o identificador do utilizador, o seu desempenho e destreza, a dificuldade da pergunta, etc.

- **Motor de regras**: O motor de regras consiste num conjunto de pares de **condição-ação**, ou seja, regras. Cada **condição** é formada por

uma combinação (conjunções e/ou disjunções) de valores definidos, correspondentes às variáveis que são recebidas no padrão. Assim que uma condição é satisfeita, isto é, que tenha valores iguais ao recebidos no padrão, a respetiva **ação** é executada, o que corresponde à geração do dialogo. Após a informação da mensagem-padrão ser devidamente analisada e estruturada, são realizados vários testes contra o motor de regras do sistema, comparando os valores obtidos com as regras estipuladas.

- **Seleção de diálogo**: Cada regra está associada a um tipo de situação/contexto possível de ocorrer durante o diálogo entre o sistema e o utilizador. Assim que uma regra é accionada, é gerado e selecionado um conjunto de frases e respostas, disponibilizando-se assim um diálogo de acordo com o contexto do momento.

3.2 Configuração do padrão

Todos as componentes do sistema *Leonardo* comunicam entre si através do envio e receção de mensagens-padrão, em formato ***JSON***. A componente de gestão de diálogos do sistema espera, por isso, receber um padrão com os seguintes campos:

- ***userid:*** identificador do utilizador que está a comunicar com o sistema, através do *chat* da plataforma;
- ***language:*** identificador do idioma do diálogo a utilizar;
- ***domain:*** identificador do domínio sob estudo, naquele momento;
- ***subdomain:*** identificador do subdomínio sob estudo, naquele momento;
- ***answer:*** indicador em relação à resposta dada pelo utilizador. `0 (incorreta), 1 (correta)`.
- ***question_lvl:*** nível de dificuldade da pergunta respondida pelo utilizador. Este nível varia de 1 a 5, por orde crescente de dificuldade;

  student_lvl: nível de avaliação geral do utilizador. Este nível varia de 1 a 5, por ordem crescente de desempenho. Os valores numéricos traduzem-se para as seguintes notas de avaliação académicas: `A-1, B-2, C-3, D-4 e E-5`;

  state: nome do processo ocupante, isto é, componente do sistema a executar no momento. Assim, quando uma componente está a processar um determinado padrão, mais nenhuma outra componente o pode manipular simultaneamente.

skill_domain: nível de destreza do utilizador, na realização de testes pertencentes ao domínio em questão. Este nível varia entre 1 a 5, por ordem crescente de destreza. Estes intervalos traduzem-se nas seguintes variáveis, para posterior análise no motor de regras: `1-'TERRIBLE', 2-'BAD', 3-'AVERAGE', 4-'GOOD', 5-'EXCELLENT'`;

performance_domain: nível de desempenho do utilizador, na realização de testes pertencentes ao domínio em questão. Este nível varia entre 1 a 5, por ordem crescente de desempenho. Estes intervalos traduzem-se nas seguintes variáveis, para posterior análise no motor de regras: `1-'TERRIBLE', 2-'BAD', 3-'AVERAGE', 4-'GOOD', 5-'EXCELLENT'.`;

skill_subdomain: esta variável tem as mesmas características que a *skill_domain*, porém é referente ao subdomínio em questão;

performance_subdomain: esta variável tem as mesmas características que a *performance_domain*, porém é referente ao subdomínio em questão;

time: nível do tempo de resposta, traduzindo o tempo, em segundos, que o utilizador demorou a responder à pergunta naquele instante. Este nível varia de 1 a 5, por ordem crescente de demora no tempo de resposta e pode ser traduzido as seguintes variáveis: `1-'SOON', 2 & 3-'GOOD', 4-'BAD', 5-'TERRIBLE'`;

typeQ: identificador do tipo de diálogo que deve ser devolvido pelo sistema. Cabe ao ***Manager*** alterar este campo no padrão, em cada iteração, para indicar o tipo de diálogo esperado pelo sistema, quando envia para a componente *Dialogue*. Os vários tipos de diálogo são expostos na secção seguinte (3.3).

Um exemplo da estrutura de uma mensagem-padrão seria:

```
{
'userid': '1',
'language': '1',
'domain': '1',
'subdomain': '1',
'answer': '1',
'question_lvl': '3',
'student_lvl': '4',
'state': '123456',
'skill_domain': '4',
'performance_domain': '4',
'skill_subdomain': '3',
'performance_subdomain': '4',
'time': '3',
'typeQ': 'greetingsA'
}
```

Após a receção de um padrão, a componente *Dialogue* irá processar todos os valores recebidos e convertê-los dentro dos intervalos descritos, de maneira a transformar os dados do padrão em informação mais sucinta e simplificada, de maneira a ser testada nas condições do motor de regras de forma mais eficiente.

3.3 Tipos de diálogo - *typeQ*

Existem vários tipos de diálogo que podem ser utilizados, consoante o contexto da conversa, entre o sistema e o utilizador. Situações diferentes requerem tipos de frases diferentes, daí terem sido elaborados os seguintes tipos de diálogos:

1. **greetingsI:** saudações iniciais, para a primeira vez que o utilizador abre o *chat* na plataforma do sistema;

2. **greetingsA:** saudações para quando o utilizador abre novamente o *chat* na plataforma do sistema;

3. **greetingsT[Soon/Late]:** saudações específicas para um determinado período de tempo. Ao entrar novamente no *chat*, é verificada a última vez em que o utilizador usufruiu desta funcionalidade na plataforma do sistema. Se se verificar que a última vez foi cerca de 5 a 60 minutos atrás (*Soon*) ou uma semana (7 dias) atrás (*Late*), nesta nova vez será enviado um diálogo de acordo com o período de tempo que passou desde a última vez;

4. **doubt:** diálogo para verificar se o utilizador deseja continuar o treino de avaliação, após ter indicado uma opção que faz duvidar o sistema;

5. **farewell:** despedidas para quando o utilizador deseja sair do treino de avaliação ou o mesmo é terminado. Neste último caso, tem-se em conta o desempenho geral do utilizador, até ao momento;

6. **domain:** diálogo para apresentar as opções de domínio/tema, para o treino de avaliação a realizar;

7. **subdomain:** diálogo para apresentar as opções de sub-domínio/sub-tema, para o treino de avaliação a realizar;

8. **time:** diálogo para apresentar quando o utilizador é demasiado rápido ou demasiando lento a responder, dependendo de vários fatores, principalmente da sua *destreza*;

9. **answer:** diálogo para apresentar quando o utilizador responde a uma pergunta. Existem várias frases especificos, de acordo com a validade da resposta e o nível de dificuldade da pergunta, para além de outros fatores, como o nível do aluno.

Por forma a dar uma **personalidade** ao sistema *Leonardo*, decidiu-se adicionar um **nível de ironia** aos diálogos. Poderá se dizer que é o *"estado de espírito"* do *Leonardo*. Assim, este nível varia entre 1 a 5, decidindo-se dividir as frases de cada tipo de diálogo, nos seguintes **tipos de frases:** `1-Incentive, 2-Normal, 3-Serious, 4-Funny, 5-Mock`.

3.4 Ligação à base de dados

A componente *Dialogue* está ligada a uma base de dados ***MongoDB***, onde se conecta a várias coleções por forma a armazenar e obter os dados relevantes para o seu funcionamento. Eis as coleções necessárias:

- ***dialog:*** coleção que armazena os diálogos genéricos, isto é, que ocorrem fora do treino de avaliação, tais como saudações (`greetings`), despedidas (`farewell`), seleção de domínio (`domain`) e confirmação (`doubt`);

- ***domain_generic:*** coleção que armazena os diálogos para quando está a decorrer um treino de avaliação, tais como seleção do sub-domínio (`subdomain`), tempos de resposta (`time`) e validade da resposta (`answer`). Estes diálogos são genéricos, no sentido que podem ser utilizados como *template* para qualquer domínio que seja criado numa fase posterior;

- ***domain_BD:*** coleção que utiliza a *template* anterior, tendo sido ainda adicionadas frases específicas sobre o tema de Base de Dados;

- ***synonyms:*** coleção que guarda um dicionário de sinónimos, criado pelo grupo do projeto, específico para as frases dos diálogos armazenados. Deste modo, confere-se uma maior diversificação na geração de frases;

- ***userHist:*** coleção desenvolvida apenas para auxiliar o desenvolvimento da componente de forma autónoma. Armazena o *timestamp* da última vez que cada utilizador utilizou o *chat*, porém, mais tarde, espera-se obter essa informação pela base de dados principal do sistema *Leonardo* ou pela componente *Manager*. Consequentemente, esta coleçao deixa de ser necessária.

Armazenamento de diálogos

Tendo em consideração todos os pontos discutidos nas duas secções anteriores, eis a estrutura genérica para o armazenamento de diálogos, nas coleções indicadas. Como estas coleções estão guardadas em *MongoDB*, recorreu-se ao formato *JSON* para elaborar esta estrutura:

```
{

'Tipo de diálogo (typeQ)': {
      'Tipo de frase':{
            'Frases': [ ... ] ,

            'Respostas':{
                  'Remete para diálogo de domínio (domain)': [ ... ] ,

                  'Remete para saída (farewell)': [ ... ] ,

                  'Remete para confirmação (doubt)': [ ... ]

                  }
            }
      }
}
```

Como se pode observar, os símbolos [**...**] representam uma lista de frases, relacionadas com as *tags* onde pertencem. Cada conjunto de frases irá servir como um *dataset* inicial para a construção do diálogo, isto é, na geração automática e dinâmica de frases, explicada na **secção 3.5**. É também apresentado um conjunto de respostas, sendo estas opções disponibilizadas ao utilizador, para permiti-lo selecionar e assim interagir com o sistema.

Diálogo suportado pelo sistema (exemplo)

Um exemplo de um diálogo entre um utilizador e o sistema *Leonardo* seria:

```
{

'greetingsI': {
      'Normal':{
            'Phrases': [ 'Então, és novo por aqui? Vamos ao trabalho!','Então, tudo bem? Vamos a isto! ☺','Olá, tudo bem? Vai um teste?', ... ] ,

            'Answers':{
                  'domain': [ 'Sim, estou preparado(a)!','Sim, vamos lá!','Estou preparado(a) para tudo!', ... ] ,

                  'farewell': [ ['Afinal, não quero estar aqui.','Não estou preparado(a).','Acho que preciso de mais tempo.', ... ] ] ,

                  'doubt': [ 'Não tenho a certeza se quero continuar...','Tem de ser, não é?','Que remédio, Leonardo!', ... ]
```

```
            }
        }
    }
}
```

Leonardo: 'Bom dia. Então, és novo por aqui? Vamos ao trabalho!'
Utilizador: 'Sim, vamos lá!'
Leonardo: 'Escolhe o tema para avaliação'

...

* Utilizador seleciona domínio e sub-domínio, realiza um treino de avaliação e termina com um bom desempenho*

...

Leonardo: 'Bom trabalho!'

3.5 Módulos da componente

Por sua vez, a nossa componente de gestão de diálogos pode ser fragmentada nos seguintes módulos:

Dialog Agent

Este é o módulo principal que dá face ao componente do gestor de diálogos. Este elemento apoia-se nos restantes módulos, descritos de seguida, para realizar as três etapas mencionadas: **análise do padrão, motor de regras e seleção de diálogo**.

Rules Engine

Este módulo implementa o motor de regras construído pela equipa de projeto, específico para este gestor de diálogos, nomeadamente o conjunto de regras para a geração e seleção de um certo tipo de diálogo. O seu funcionamento geral é explicado na **secção 3.1**. Uma regra é então um par de *condição - ação*, tendo em conta as variáveis do padrão. Pode-se afirmar, portanto, que o motor de regras pode ser traduzido por uma **árvore de decisão**:

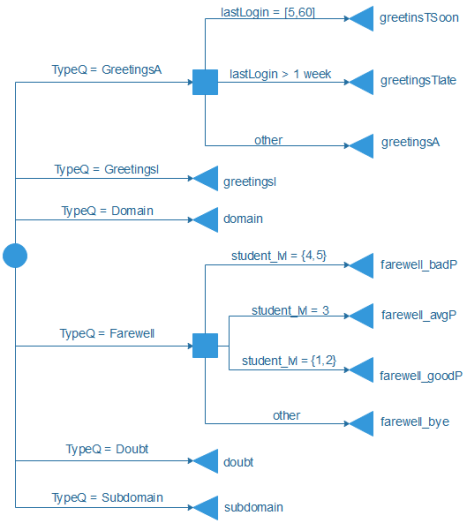

Figura 5. Extrato da árvore de decisão que traduz o motor de regras da componente

Uma árvore de decisão é um modelo usado de inferência indutiva, prático e efeciente. Cada ramo da árvore representa uma decisão e cada nodo uma condição. Já as folhas da árvore apresentam uma conclusão. Assim, no contexto do gestor de diálogos, cada ramo é uma variável do padrão com um certo valor, os nós são as condiçoes, terminando nas folhas que representam um tipo de diálogo. Todos os caminhos possíveis da árvore de decisão formam as regras do motor de regras utilizado.

Um exemplo de uma regra seria:

```
SE typeQ = 'answer' & answer = '0' & question_lvl =  '1' ||
'2' || '3' & student_lvl = '1' || '2' ENTÃO choose_dialog(
list_answer_wrong_easy , ['Mock','Serious'])
```

Analisando cada campo desta regra, podemos concluir que quando o utilizador responde a uma pergunta (**typeQ = 'answer'**), a resposta está incorreta (**answer = '0'**), porém a pergunta é fácil (**question_lvl = 1 ou 2 ou 3**) e o utilizador é um bom aluno (**student_lvl = 1 ou 2**), então o diálogo gerado terá como base as frases criadas especificamente para essa situação, num tom **sério ou jocoso**, pois foi considerado o mais adequado para este contexto.

Collector

O módulo *Collector* é responsável por fazer a ligação entre a componente e a base de dados *MongoDB*. Tem, assim, a função de armazenar e obter os dados necessários para a gestão do diálogo do sistema, como: fazer conexão às coleções apropriadas, obter frases dos diálogos guardados, obter os sinónimos e recolher informação do utilizador, como o seu nome e *timestamp* da última vez que utilizou o *chat*.

Generator

O módulo *Generator* confere a função de gerar frases automaticamente e dinamicamente, à componente *Dialogue*, para o diálogo entre o sistema e o utilizador. É assim concedida uma maior versatilidade e flexibilidade no diálogo do *Leonardo*, evitando a repetição e parecendo mais "humano".

Este gerador utiliza o conceito das cadeias de *Markov* para formar as frases. As cadeias de *Markov* são úteis para prever o estado futuro com base nas características do estado presente. Assim , o programa utiliza e analisa as frases de um *dataset* inicial e, a partir daí, tenta prever e gerar novas frases. Este *dataset* inicial é constituído pelo conjunto de frases pré-definidas, referidos na secção 3.4, guardados em *MongoDB*. É, portanto, uma geração de frases **supervisionada**.

Utiliza-se, assim, as cadeiras de *Markov* para prever quais as sequências de palavras mais adequadas que precedem a um outro conjunto de palavras.

De uma forma mais específica, o programa cria um dicionário com as palavras, recebidas das frases iniciais do *dataset*, e a palavra que vem a seguir destas, aplicando, posteriormente, as técnicas das cadeias de *Marcov* sobre essa informação, tentando formar frases que façam sentido. São geradas cerca de 50 frases diferentes de cada vez que é executado algum pedido de diálogo e selecionada uma aleatóriamente, para dar oportunidade de as técnicas de cadeias de *Markov* fazerem efeito.

É por esta razão que as cadeias de *Markov* são bastante utilizadas neste tipo de situações, nomeadamente *chatbots*.

Auxiliary

Este módulo, como o seu nome indica, reúne todas as funções auxiliares que suportam os restantes módulos deste componente. É aqui que estão os métodos de *parse* do padrão *JSON* recebido, a seleção do diálogo e a substituição de palavras/sinónimos na geração de frases.

4 Recursos Computacionais

Todos as componentes do projeto *Leonardo* estão a ser desenvolvidos na linguagem de programação ***Python*** [3]. A escolha para a utilização desta linguagem apoia-se no facto de esta ser uma linguagem fácil de utilizar e manipular, para

além de fornecer um vasto leque de funcionalidades e opções, o que contribui para um sistema mais eficiente e complexo. Visto que as componentes iniciais e principais já estavam a ser produzidas em *Python*, decidiu-se também fazer a componente *Dialogue* com a mesma, de modo a facilitar a sua integração no projeto. É ainda de mencionar que a componente *Dialogue* foi completamente desenvolvida e testada num ambiente *Windows 10* [1], que possuía todas as ferramentas instaladas necessárias à sua concepção.

Entre todos os recursos de programação utilizados na conceção desta componente, salientam-se os seguintes: **JSON, MongoDB, NLTK e PyKnow**.

Como referido anteriormente, o formato JSON [4] serviu para estruturar não só as mensagens-padrão utilizadas na comunicação entre as diferentes componentes do sistema, como também para traçar a estrutura das coleções em MongoDB, de forma a armazenar diálogos, sinónimos e informações sobre os utilizadores.

O *software* MongoDB [5] é utilizado para armazenar as coleções que guardam os dados necessário para o funcionamento do sistema. Para conectar o programa da componente a MongoDB recorreu-se à *API* ***pymongo***, uma biblioteca que permite fazer a ligação à base de dados e executar *queries NoSQL*, através de código *python*.

Já no módulo *generator* é importado o pacote *NLTK (Natural Language Toolkit)* [6], para ajudar no processamento de palavras e geração de frases.

Por fim, e não menos importante, é utilizada a biblioteca *PyKnow* [7] para formar o motor de regras da componente. O *PyKnow* é uma biblioteca *Python* para criar *expert systems* fortemente inspirada por CLIPS. É através dos métodos que disponibiliza que é possível traduzir a árvore de decisão elaborada, em código *Python*.

Figura 6. Recursos computacionais utilizados para a componente *Dialogue*, em *Python*

[1] www.microsoft.com/pt-pt/store/b/windows

Em relação a este último, o motor de regras definido com *PyKnow* pode ser resumido nos seguintes passos:

1. Recebido o padrão e convertido os seus valores, declara-se um **Facto** com as suas variáveis:

   Facto → *Pattern*(*username* = pattern[0], *language* = pattern[1], *domain* = pattern[2] ,*subdomain* = pattern[3], *answer* = pattern[4], *question_lvl* = pattern[5], *student_lvl* = pattern[6], *state* = pattern[7], *skill_domain* = pattern[8], *performance_domain* = pattern[9], *skill_subdomain* = pattern[10], *performance_subdomain* = pattern[11], *time* = pattern[12], *typeQ* = pattern[13])
   O facto fica guardado na base de conhecimento do motor.

2. Em segundo lugar, corre-se o motor para que este tome conhecimento dos factos existentes na sua base de conhecimento e testa cada um deles contra as regras definidas préviamente nele. Este teste consiste em verificar a existência de uma condição da regra, que corresponda a um mesmo facto declarado na base de conhecimento. Por exemplo:

   Regra → @*Rule*(*Pattern*(*typeQ* = 'time', *time* = '4', *skill_subdomain*=L('2') | L('1'))
   def doubt(*self*) : .....

   Se o Facto declarado anteriormente tiver *typeQ = 'time', time = '4'* e *skill_subdomain = 1 ou 2*, então a regra é disparada, sendo o método associado executado (*doubt*).

3. Por fim, cada método associado a uma regra tem o objetivo de selecionar o diálogo correspondente à situação que a regra apresenta. Após gerado e selecionado, as frases são devolvidas ao utilizador, pelo sistema.

5 Instalação e Configuração

Para replicar as condições em que o grupo desenvolveu o projeto, e de forma a que funcione em qualquer máquina, apresentam-se e descrevem-se as condições de instalação e configurações base, enumerando os recursos necessários.

Começando pelo **MongoDB**, foi utilizada a versão 3.4.10. Para que o sistema consiga aceder corretamente à base de dados, de forma a gerar as frases automaticamente, é necessário importar as coleções *Dialog*, *DomainBD*, *Domain_generic*, *Synonyms* e *UserHist*. Como referido anteriormente, na secção 3.4, primeira coleção contém todos os tipos de diálogos considerados gerais, i.e., que não envolvem um contexto específico, como quando um aluno seleciona a temática "Base de Dados". Para tal, surge a coleção *DomainBD*, que descreve as frases típicas de BD, com palavreado típico da área. A terceira coleção é, também, relativa à parte do teste a partir da escolha do tema de avaliação, mas é genérica, servindo de *template* para um outro tema como, por exemplo, Matemática. A quarta coleção envolve sinónimos que permitem enriquecer e variar o diálogo fornecido ao utilizador, e a *UserHist* possui alguns dados de cada utilizador, como a *timestamp* da última vez que o mesmo entrou no sistema. Dados como o seu desempenho geral ou destreza em determinados temas não são necessários estarem nesta coleção, pois provém do padrão.

Quanto à linguagem de desenvolvimento - *Python* -, a versão utilizada foi a 3.7.3. Utilizou-se a biblioteca *PyKnow* para geração de regras (que permitem escolher os diálogos adequados na base de dados), e a biblioteca *pymongo* para a conexão à BD.

Por fim, quanto ao NLTK (*Natural Language Toolkit*), biblioteca utilizada pelo grupo para processamento de linguagem natural, utilizaram-se as suas sub bibliotecas *Tokenize* e *Treebank* que permitem a manipulação de *strings*.

6 Funcionamento do Sistema

Passando para a componente específica desenvolvida pelo grupo - *Dialogue* - apresenta-se, de seguida, um esquema detalhado de forma a abranger as diversas etapas do seu funcionamento. Relembre-se que estaa componente apenas interatua com o *Manager*, sendo que é este último que envia sempre um *input* e recebe o *output* correspondente.

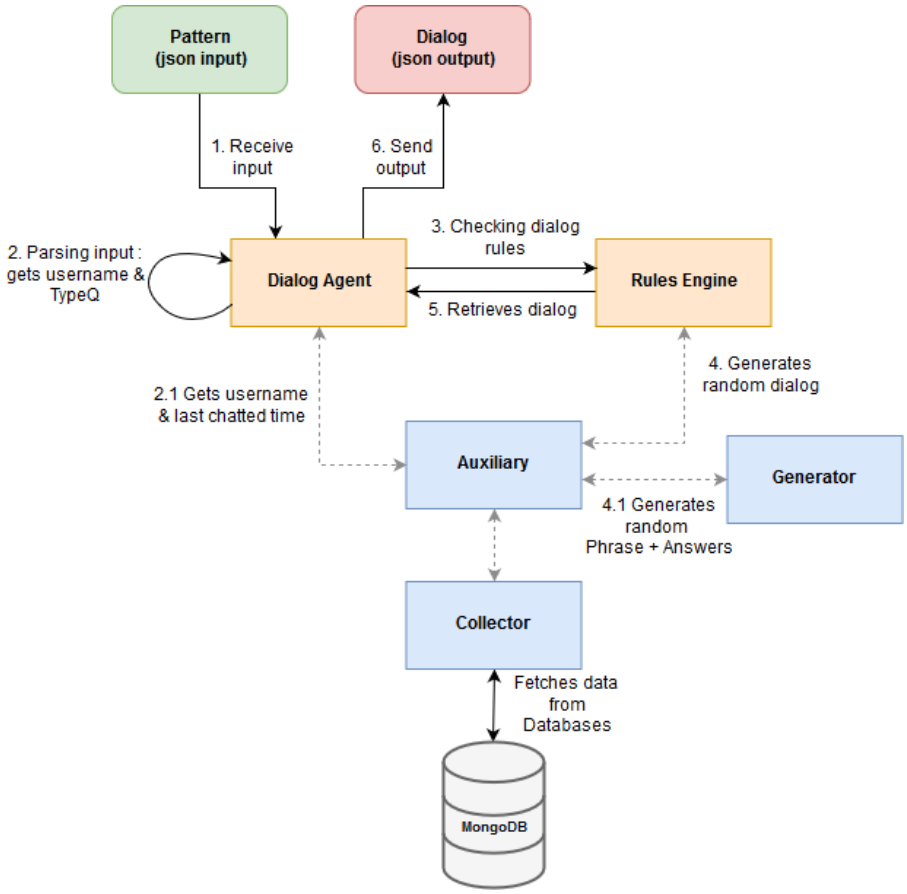

Figura 7. Esquematização da componente *Dialogue*

Assim, numa primeira fase, a componente recebe um padrão (**passo 1**), *input* sob a forma de vetor, onde cada posição é indicativa de uma característica.

['1', '1', '1', '1', '1', '3', '4', '123456', '4', '4', '3', '4', '3', '']

Mais especificamente,

```
{
'userid': '1',
'language': '1',
'domain': '1',
'subdomain': '1',
'answer': '1',
'question_lvl': '3',
'student_lvl': '4',
'state': '123456',
'skill_domain': '4',
'performance_domain': '4',
'skill_subdomain': '3',
'performance_subdomain': '4',
'time': '3',
'typeQ': ''
}
```

Este padrão é recebido pelo módulo *Dialog Agent*, que vai analisar este *input*, começando por obter da base de dados (**passo 2**) o nome de utilizador e o tipo de diálogo atual, ao extrair os valores de *userid* e *typeQ*. No caso de um utilizador que acabe de chegar ao sistema, o tipo de diálogo será de saudação, dependendo de há quanto tempo o mesmo não entra em contacto com o *Leonardo*. Se já não for a primeira vez que o utilizador entra na plataforma, já existirão dados relativos à sua *performance* ou *skill*, pois este já realizou testes. É de notar que sempre que existe um acesso à base de dados, este é feito pelo módulo *Collector*, que é a única que comunica com a parte de permanência dos dados.

Obtidos o nome de utilizador e tipo de diálogo, por exemplo, para um aluno que entrou no sistema e já não o fazia há, pelo menos, uma semana, o *input* ficará na forma:

```
{
'userid': '1',
'language': '1',
'domain': '1',
'subdomain': '1',
'answer': '1',
'question_lvl': '3',
'student_lvl': '4',
'state': '123456',
'skill_domain': '4',
'performance_domain': '4',
'skill_subdomain': '3',
'performance_subdomain': '4',
'time': '3',
'typeQ': 'greetingsTLate'
}
```

De seguida, continuando para o **passo 3** do esquema do início da secção, observa-se que o padrão é enviado para o módulo *Rules Engine*. Aqui são geradas as regras conforme o tipo da questão, e em certos casos, com base no desempenho do utilizador. A regra escolhida determina a coleção e diálogo da mesma a usar para gerar frases (**passos 4 e 4.1**). Uma vez a coleção acedida e a frase gerada automaticamente pelo módulo *Generator*, a mesma é transmitida do *Rules Engine* de volta para o *Dialog Agent* (**passo 5**), que devolve, finalmente, ao *Manager* o diálogo a mostrar ao estudante (**passo 6**).

Considerando o mesmo exemplo, um aluno que não é um novo membro no sistema e que não entra no mesmo há mais de uma semana, receberá algo como o seguinte diálogo e opções corespondentes, das quais ele escolherá uma para continuar para o teste, sair do sistema e de incerteza:

Leonardo: 'Então, foste de férias? Vamos lá praticar!' ☺
User* - opção *domain: 'Bora lá!'
User* - opção *farewell: 'Preciso de estudar. Adeus'
User* - opção *doubt: 'Tem mesmo que ser?'

Todo este raciocínio repete-se enquanto um utilizador estiver no sistema, desde o momento que chega, passa pela escolha do tema do treino de avaliação, até à realização do mesmo e consequente saída. Durante a realização do treino, o *Manager* nem sempre pede à componente *Dialogue* que intervenha, mas, caso peça, serão geradas frases conforme o aluno acerte ou erre na pergunta, e conforme a pergunta seja fácil ou difícil. Aqui, a pergunta pode ter um cariz mais impessoal, ou, por outro lado, possuir um "gozo" subjacente.

7 Conclusões e Trabalho Futuro

A componente desenvolvida e exposta neste documento pode ser considerada essencial para tornar o sistema *Leonardo* mais interativo e cativante, dando-lhe uma *"personalidade"*, visto que este é capaz de se adaptar às diferentes etapas que o utilizador atravessa de uma forma não repetitiva e natural.

A capacidade que este componente tem de improvisar o diálogo com o utilizador é um ponto positivo, porém, este improviso apenas é possível devido ao facto de o componente ter por base um conjunto de frases que foram geradas manualmente. Este ponto, por um lado, é positivo, uma vez que permite garantir que o componente é capaz de criar um diálogo no qual não existem dúvidas sobre o seu objetivo, evitando assim a probabilidade de o componente gerar algo que não é pretendido. Por outro lado, é negativo, visto que se o conjunto de frases não apresentar diversidade suficiente pode fazer com que o Leonardo se torne repetitivo e pouco estimulante para o utilizador.

Com o objetivo de melhorar este componente, no futuro, poderia ser adicionado um maior número de variáveis ao padrão que é recebido pelo componente,

o que permitiria melhorar o motor de regras e assim obter diálogos mais adaptados ao contexto e ao utilizador - uma *personalidade* mais refinada. Um exemplo de uma possível variável seria o nível de *sensibilidade* do utilizador.

Referências

1. LiveChat Inc, 2019 *ChatBot - Automate customer service with ChatBot*, viewed 21 February 2019
<www.chatbot.com>

2. Luis, Severin , A & H 2019 *Duolingo - Aprenda idiomas de graça.*, viewed 20 February 2019
<https://pt.duolingo.com/>

3. Python Software Foundation, 2019 *Python is powerful... and fast*, viewed 1 March 2019
<www.python.org>

4. Douglas Crockford, 2019 *JSON - Introducing JSON*, viewed 1 March 2019
<www.json.org>

5. MongoDB Inc, 2019 *MongoDB - The database for modern applications*, viewed 1 March 2019
<www.mongodb.com>

6. NLTK Project, 2019 *NLTK - Natural Language Toolkit*, viewed 12 March 2019
<www.nltk.org>

7. buguroo - offensive security, 2019 *PyKnow - Expert Systems for Python*, viewed 12 March 2019
<pyknow.readthedocs.io/en/stable/>
<https://github.com/buguroo/pyknow>