Chatbot para serviços bancários

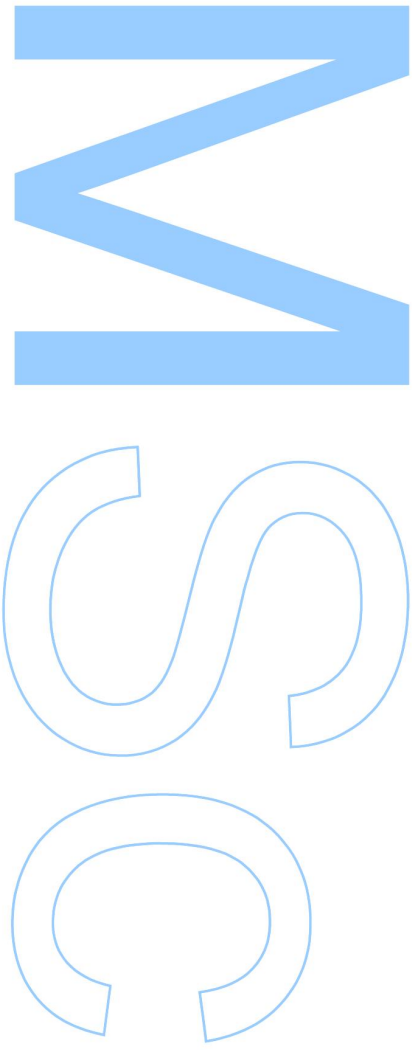
Fábio André Alves Teixeira

Mestrado Integrado em Engenharia de Redes e Sistemas Informáticos

Departamento de Ciência de Computadores
2018

Orientador

Alípio Mário Guedes Jorge, Professor Associado, Faculdade de Ciências da Universidade do Porto



Dedico à minha família e namorada

Abstract

Nowadays, people already want to stop going to the supermarket to do their shopping, since they buy most of their products online, such as clothes, shoes, gadgets, etc. There is, therefore, a clear need to avoid work and extra time that can be reused in another way. In this sense, chatbots arise. These programs, which try to simulate human conversations, are completely revolutionizing the methodology of business operation, ensuring a closer interaction with users and allowing a simpler delivery of their products.

In the banking area, the scenario is no different. If the application of ATMs was already revolutionary, the possibility of doing the same actions on a smartphone or computer took the paradigm to an easier level of adhesion for the customer. Imagine, now, having the best of both worlds: the independent hypothesis of carrying out the planned actions 24/7, since there is no obligation to respect the working hours of the company's employees, and, at the same time, having a total and specialized service, which ensures the customer a greater satisfaction.

In this dissertation, we describe a chatbot prototype developed to respond to various banking services, from creating an account to making payments, allowing a simple and direct interaction with the user through a series of generalization conversations. Three different approaches were used, the first through the Python `Chatterbot` library, in conjunction with a dialog dataset; the second by the manual creation of intents and use of NLTK tools; and finally, the third and chosen one focused on an API for classification (LUIS, from Microsoft).

The goal was not only to have a way of finding out what the intent behind each phrase was, but also to ensure robustness to variations in the use of names. That is, different entities (such as Amount or Recipient) can receive all kinds of values and names and not just a set of them preprocessed. As the first two approaches did not meet the requirements, the solution found was to use the Microsoft LUIS API to discover the intent and key-values of phrases dynamically. In this way, the objectives were fulfilled.

Keywords. Chatbot, Chatterbot, Python, Natural Language Processing, Machine Learning, Banking, LUIS, Intents, Entities.

Resumo

Nos dias de hoje, as pessoas já querem deixar de se deslocar ao supermercado para fazer as suas compras, como também adquirem grande parte dos seus produtos via *online*, tais como roupa, sapatos, *gadgets*, por aí fora. Há, portanto, uma clara necessidade de evitar trabalho e tempo extra que podem ser reaproveitados de outra forma. Nesse sentido, surgem os *chatbots* – programas que tentam simular conversa humana –, que estão a revolucionar por completo a metodologia de funcionamento das empresas, garantindo uma interação mais próxima com os utilizadores e permitindo uma maior simplicidade na oferta dos seus produtos.

Na área bancária o cenário não é diferente. Se a aplicação de ATMs já foi revolucionária, a possibilidade de efetuar as mesmas ações sob um *smartphone* ou computador levou o paradigma para um nível de adesão mais facilitado para o cliente. Imagine-se, agora, ter o melhor de dois mundos: a hipótese independente de realizar as ações pretendidas 24/7 já que não há a obrigação de respeitar horários de trabalhos dos funcionários das empresas, e ao mesmo tempo ter um acompanhamento total, integral e especializado, que permite ao cliente uma maior satisfação pelo serviço.

Nesta Dissertação, descrevemos um protótipo de um *chatbot* desenvolvido para responder a diversos serviços bancários, desde a criação de uma conta até à realização de transferências ou pagamentos, permitindo uma interação simples e direta com o utilizador, através de uma série de conversas de generalização. Foram utilizadas três abordagens diferentes, a primeira através da biblioteca `Chatterbot` do Python, em conjunto com um *dataset* de diálogos; a segunda pela criação manual de *intents* e uso de ferramentas do NLTK; e finalmente a terceira (e a escolhida) centrou-se na utilização de uma API para classificação (LUIS, da Microsoft).

O objetivo era não só ter forma de descobrir qual era a intenção por trás de cada frase que o *chatbot* recebia, como também garantir robustez a variações no uso de nomes. Isto é, diferentes entidades (como o Montante ou o Destinatário) poderem receber todo o tipo de valores e nomes e não apenas um conjunto deles pré-processados. Como as duas primeiras abordagens não foram capazes de exercer o propósito, a solução encontrada foi a utilização da API LUIS, da Microsoft, para classificar todas as frases e descobrir a intenção e os valores-chave das mesmas. Desta forma, os objetivos foram cumpridos.

Keywords. Chatbot, Chatterbot, Python, Natural Language Processing, Machine Learning, Banking, LUIS, Intents, Entities.

Agradecimentos

Para que este projeto fosse bem-sucedido, o apoio e incentivo de diversas pessoas e entidades foram imprescindíveis, pelo que fica o agradecimento:

Ao Prof. Alípio Jorge, pela orientação, atenção, disponibilidade, sensatez e valor das suas opiniões;

Ao DCC, FCUP e respetivos docentes e funcionários, pela constante aprendizagem e evolução tanto a nível de conteúdos escolares como pessoais nos últimos 5 anos;

À minha família, pelas condições e possibilidade de, dia após dia, trabalhar sem interrupções;

À Cláudia Almeida, pelo apoio, ajuda e carácter e incentivação para a conclusão do trabalho.

Obrigado a todos.

Conteúdo

Lista de Tabelas

Lista de Figuras

Lista de Blocos de Código

Acrónimos

AIML Artificial Intelligence Markup Language

ATM Automatic Teller Machine

BD Base de Dados

DCC Departamento de Ciência de Computadores

FCUP Faculdade de Ciências da Universidade do Porto

IA Inteligência Artificial

IoT Internet of Things

I/O Input/Output

IT Information Technology

LUIS Language Understanding Intelligent Service

NLTK Natural Language Toolkit

NLU Natural Language Understanding

NLP Natural Language Processing

P2P Peer to Peer

SaaS Software as a Service

Capítulo 1

Introdução

Um *chatbot* é um *software* de inteligência artificial que pode simular uma conversa numa linguagem percetível ao utilizador, como o português, o inglês, etc. A tecnologia, que cada vez reúne mais interesse, tem como grande propósito a otimização da interação entre humanos e máquinas, nomeadamente para impulsionar os negócios das empresas. Do ponto de vista tecnológico, um *chatbot* representa a evolução natural de um sistema de pergunta-resposta que utiliza *Natural Language Processing*. A formulação de respostas a perguntas em linguagem natural é um dos exemplos mais típicos de NLP aplicado a *software* que as empresas apresentam aos consumidores. Hoje em dia já é frequente encontrarmos os *bots* embebidos em aplicações de *messaging*, sites ou até *apps* móveis.

Ora, como estes *softwares* aumentam, em teoria, a rapidez com que as tarefas são realizadas e diminuem o trabalho que tais tarefas requerem e também a complexidade inerente, não é de estranhar que várias envolvências do mundo dos negócios estejam a apostar na tecnologia. Uma das áreas que tem mostrado interessadas é a bancária, com diversos bancos a nível Internacional a criarem *bots* programáveis para aproximar a relação com o utilizador final, especialmente se tivermos em conta que é bastante mais fácil realizar os encargos quando há alguma conversa por trás do que quando não há interação de todo. Com isto torna-se possível realizar ações 24/7, já que não há a obrigação de respeitar horários de trabalhos dos funcionários das empresas, e ao mesmo tempo ter um acompanhamento total, integral e especializado, que permite ao cliente uma maior satisfação pelo serviço.

1.1 Motivação

Os *bots* estão em claro crescimento, mas, e como se estima que no futuro tenham um peso cada vez maior na vida das pessoas, este é um excelente momento para apostar na ideia, já que é mais fácil ter influência no mercado por diferenciação quando ainda não há sobrelotação. O facto das aplicações de *messaging* estarem a suplantar as redes sociais como principal plataforma de utilização na Internet exponencia ainda as vantagens de construir um *chatbot*.

Há, no entanto, uma grande linha que pode ditar o sucesso de um *bot*, que é a sua capacidade de se aproximar do carácter humano, seja pela sua interação, linguagem, conselhos ou até simplicidade da interface. No setor da banca, as capacidades dos *bots* e as suas *features* permitem diferentes focos de inserção, desde apoio ao cliente até pagamentos e transferências, ou até transações de Blockchain. As aplicações dos *chatbots* podem estar embutidas nos *sites* e *apps* particulares dos bancos ou nas plataformas já existentes, como o Facebook Messenger, Skype, etc. Mas o grande desafio é fazer da versatilidade de opções dos Multibancos e ATMs - um dos sistemas embutidos mais revolucionários das últimas décadas - mais *frendly-user*, ou seja, menos complexas, mais rápidas e menos trabalhosas devido à não imposição de ter de realizar deslocamentos. Com isto, pretende-se um aumento na atividade dos clientes dos bancos, com melhorias financeiras para as entidades e com uma maior interação e aproximação entre utilizador final e o fornecedor do serviço.

Todos estes fatores serviram como motivação para o desenvolvimento do trabalho. A possibilidade de ter tecnologia que assume um carácter inovador na área, que está disponível 24 por 7, que é um potenciador de propostas negócios para empresas, que fornece satisfação aos utilizadores e que otimiza uma panóplia de distintas tarefas, é uma oportunidade irrecusável.

1.2 Objetivos

Os objetivos do trabalho centraram-se em:

- Analisar o estado corrente da tecnologia e o seu passado recente;
- Analisar ferramentas existentes para possível adaptação futura.
- Produzir um protótipo demonstrável tão sofisticado quanto possível.

As funcionalidades pretendidas para o *bot* são as seguintes:

- Capacidade de gerar conversa de generalização, como mensagens de boas-vindas, de agradecimento ou de despedida;
- Possibilidade de criar ou abrir contas, de tipo poupança ou corrente, para utilizadores autenticados com nome;
- Criação de entradas numa base de dados para simular os dados das contas dos utilizadores;
- Possibilidade de efetuar ações sobre os dados bancários, como transferir, levantar ou depositar dinheiro;
- Capacidade para a qualquer momento aceder ao saldo da conta.

Para a resolução destas tarefas e cumprimento dos objetivos definidos, foram estudadas três abordagens, tendo uma delas sido considerada a mais adequada. Esta consiste numa solução desenvolvida neste trabalho, que utiliza um classificador de intenções do *bot framework* (LUIS) da Microsoft. Para a representação interna estruturada das mensagens em língua natural trocadas entre o humano e a máquina foi utilizada a notação JSON.

1.3 Resultados

O resultado final foi um protótipo de um *chatbot* bancário em língua portuguesa, de nome ChatBanco, programado em Python, com utilização de bases de dados em SQLite e ainda de um *bot framework* auxiliar.

Para além disso, o processo contou ainda com uma série de passos intermédios com um claro valor acrescentado, seja ao nível dos dados, algoritmos ou *software*, tais como:

- Mais de 450 falas de diálogo treinadas no *bot framework*;
- Cerca de 15 intenções e entidades definidas no LUIS;
- Ficheiros de texto programáveis com respostas geradas para cada intenção;
- Funções para criar uma conta, saber o saldo e depositar, levantar ou transferir dinheiro;
- Base de dados que simula os dados bancários extraídos da conversa após a criação de uma conta.

1.4 Estrutura da Tese

O presente relatório está organizado da seguinte forma: no capítulo 2 é apresentado o enquadramento dos *chatbots* no Mundo atual, sendo que é também analisado o estado corrente da tecnologia, bem como ferramentas comerciais e produtos similares existentes; no capítulo seguinte (3) é exibida a análise de requisitos, juntamente com abordagens testadas antes do modelo final, mas que não foram escolhidas para o desenvolvimento; de seguida (capítulo 4) é detalhado todo o processo de criação do *bot*, em conjunto com blocos de código e diagramais auxiliares; há ainda espaço para experiências e testes (capítulo 5); e por fim faz-se uma revisão sintética da oportunidade que representam os *chatbots*, assim como o que poderá ser o futuro do trabalho (capítulo 6).

Capítulo 2

Chatbots

Este capítulo destina-se a oferecer uma visão sobre aquilo que têm sido os *chatbots* ao longo dos tempos e o seu enquadramento no Mundo real, com particular realce para o estado atual da tecnologia.

Em 1950, a meio do século passado, Alan Turing[1] publicou o artigo *Computing Machinery and Intelligence*, no qual partilhou pela primeira vez o *Turing test*. Este teste permitia distinguir o comportamento de um programa de computador com um de um humano e acabou por se revelar altamente influenciador e um conceito fundamental na área da Inteligência Artificial[2].

A partir desta data os especialistas em *Computer Science* que queriam criar *software* inteligente começaram a ter sempre em conta o Teste de Turing, uma vez que ultrapassado o mesmo, havia, de facto, a garantia que o programa iria comportar-se como um simulador de consciência humana.

O professor Joseph Weizenbaum[2], em 1966, criou aquele que ficou conhecido como o primeiro *chatbot* da história, o ELIZA. Apesar do sucesso do *bot* e de parecer genuinamente inteligente, já que conversava como um humano, o produto de Weizenbaum não era mais do que um conjunto de respostas pré-programadas que eram emitidas sempre que alguma palavra-chave era referida e tinha apenas 204 linhas de código.

Os primeiros *bots* com inteligência apareceram apenas no século XXI e já envolviam Processamento de Linguagens Naturais[3], uma *feature* que se tornou indispensável para que estes reunissem condições de aprendizagem. O desenvolvimento da indústria ganhou, então, uma forte componente tecnológica, mas foi finalmente em 2016 que o estudo e construção dos *chatbots* sofreu um aumento de interesse brutal e tudo graças a uma mudança de paradigma que vinha a ser gradual, mas que poucos tinham reparado.

Em abril de 2016 o Facebook anunciou que o Messenger passaria a suportar conversações individuais com *bots* personalizados, o que exponenciou, de uma forma assinalável, o mediatismo

[1] Alan Mathison Turing (1912-1954) foi um matemático, lógico, criptoanalista e *computer scientist* britânico.

[2] Joseph Weizenbaum (1923-2008) foi um escritor e *computer scientist* teuto-americano. Professor emérito do MIT, também criou a linguagem de programação SLIP.

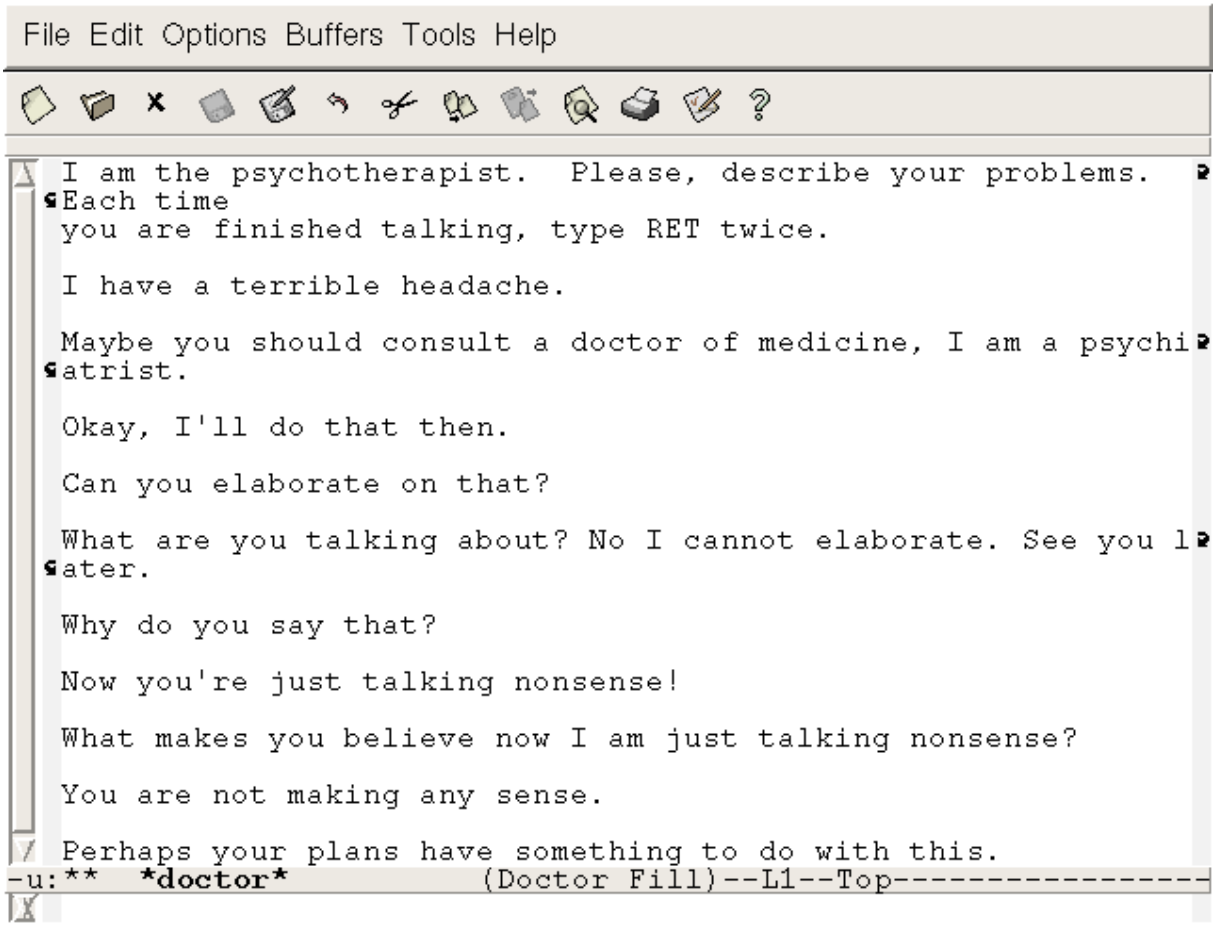


Figura 2.1: Interação com o ELIZA. Retirado de https://en.wikipedia.org/wiki/ELIZA.

destes simuladores. As empresas começaram a apostar na tecnologia, e ainda que tenham demorado algum tempo a afinar os programas (os primeiros *bots* foram um fracasso), os *chatbots* atingiram um grande nível de relevo.

"~90% do nosso tempo no telemóvel é passado em plataformas de e-mail e mensagens. Adoraria formar equipas que criassem coisas para sítios onde os consumidores vão!" - **Niko Bonatsos**, Diretor Geral na General Catalyst.

"As pessoas estão agora a passar mais tempo em aplicações de mensagens do que nas redes sociais, o que é um grande ponto de viragem. As aplicações de mensagens são as plataformas do futuro e os *bots* vão ser o meio para aceder a todos os tipos de serviço." - **Peter Rojas**, empresário e trabalhador da Betaworks.

"As mensagens são os locais onde passamos uma grande parte do tempo e esperamos comunicar. É ridículo que ainda tenhamos que ligar para a maioria dos negócios." - **Josh Elman**, sócio-parceiro na Greylock.

Ao trazer os *chatbots* para os locais onde as pessoas passavam mais tempo no telemóvel, a adesão dos mesmos sofreu um enorme *boost*, transformando de uma forma gigantesca a forma

como as empresas chegavam aos seus consumidores. Hoje em dia, outras plataformas como o Skype, o Slack ou o Telegram também já permitem o lançamento e implementação de *bots* nos seus serviços e é provável que mais ferramentas de *messaging* se juntem à lista no futuro.

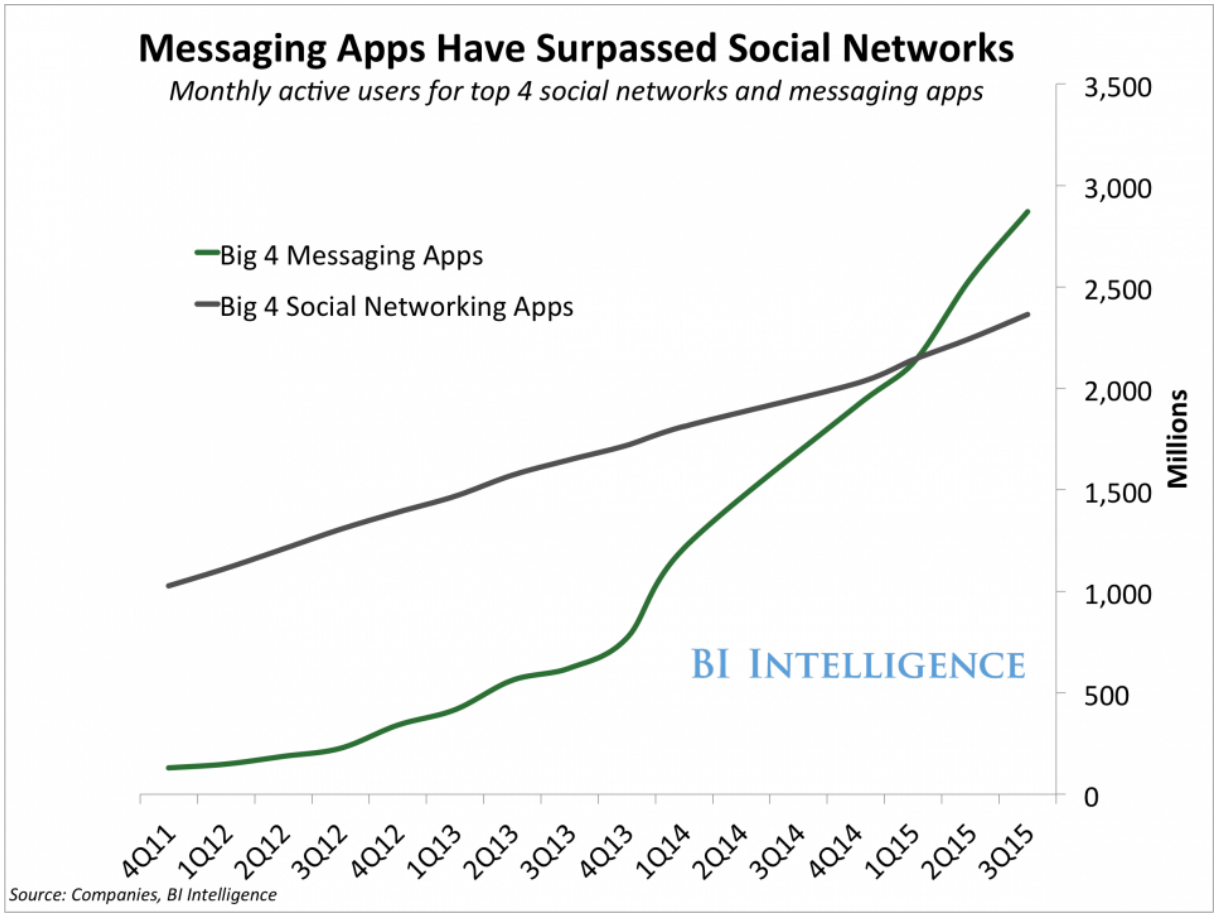


Figura 2.2: Diferenciação entre a utilização de aplicações de mensagens e redes sociais de 2011 a 2015. Fonte: *Companies, BI Intelligence*.

2.1 Estado da Arte

Os *chatbots* de hoje são desenhados para responder a propósitos muito específicos. Seja para vender produtos, fornecer uma linha de apoio ao cliente mais simplista, saber mais sobre concertos de uma banda, consultar o tempo, ler as notícias, etc. O objetivo é que forneçam um serviço mais rápido, conciso e que não obrigue as pessoas a longas pesquisas para obterem aquilo que pretendem.

Os *bots* dividem-se em duas categorias: os que detetam simplesmente *keywords* e respondem com frases pré-formuladas e os que ganham condições de aprendizagem com análises constantes das palavras trocadas ao longo do tempo, estudando a frequência com que aparecem, entre outros fatores, que os classificam como inteligência artificial.

As pequenas e grandes empresas começam a gerar *chatbots* para se aproximarem dos seus clientes e garantirem uma maior probabilidade de financiamento, por questões de acessibilidade e simplicidade, uma vez que é muito mais fácil adquirir produtos quando o processo para o fazer é minimalista de esforço.

Os *chatbots* têm, de facto, o potencial para substituir tarefas extensas e trabalhosas com um simples método de conversação e interação com sistemas designados a tais propósitos. Na área da banca, estima-se que, em 2020, os consumidores vão passar a gerir 85% de todos os seus negócios e associações com os bancos através de *chatbots* que vão funcionar e servir como assistentes financeiros pessoais[4].

2.1.1 Componentes de um *chatbot*

Para desenhar um *chatbot*, é necessário compreender e identificar as partes constituintes inerentes a estes pacotes de *software*. Assim, um *chatbot* pode ser dividido em 3 partes: *Responder*, *Classifier* e *Graphmaster*[5].

1. *Responder*
    - É a parte que desempenha o papel da interface entre o *bot* e o utilizador. As tarefas do *Responder* incluem a transferência de dados do utilizador para o *Classifier* e o controlo do I/O.
2. *Classifier*
    - É a parte entre o *Responder* e o *Graphmaster*. Esta camada filtra e normaliza o *input*, segmenta-o em componentes lógicos, transfere a frase normalizada para o *Graphmaster*, processa o *output* do *Graphmaster* e manipula as instruções de sintaxe da base de dados.
3. *Graphmaster*
    - É a parte que executa as tarefas de organizar o conteúdo ou armazenar os algoritmos de correspondência de padrões.

2.1.2 Estado da investigação

Os *chatbots* têm representado uma grande parte da investigação científica na área da tecnologia desde o início do século XXI, tendo ganho significante importância na última década. Muitos autores têm desenvolvido projetos que incidem na implementação de *bots*, apesar da área bancária continuar a parecer mais um tema "para o futuro", por toda a dificuldade que representa construir um *chatbot* bancário.

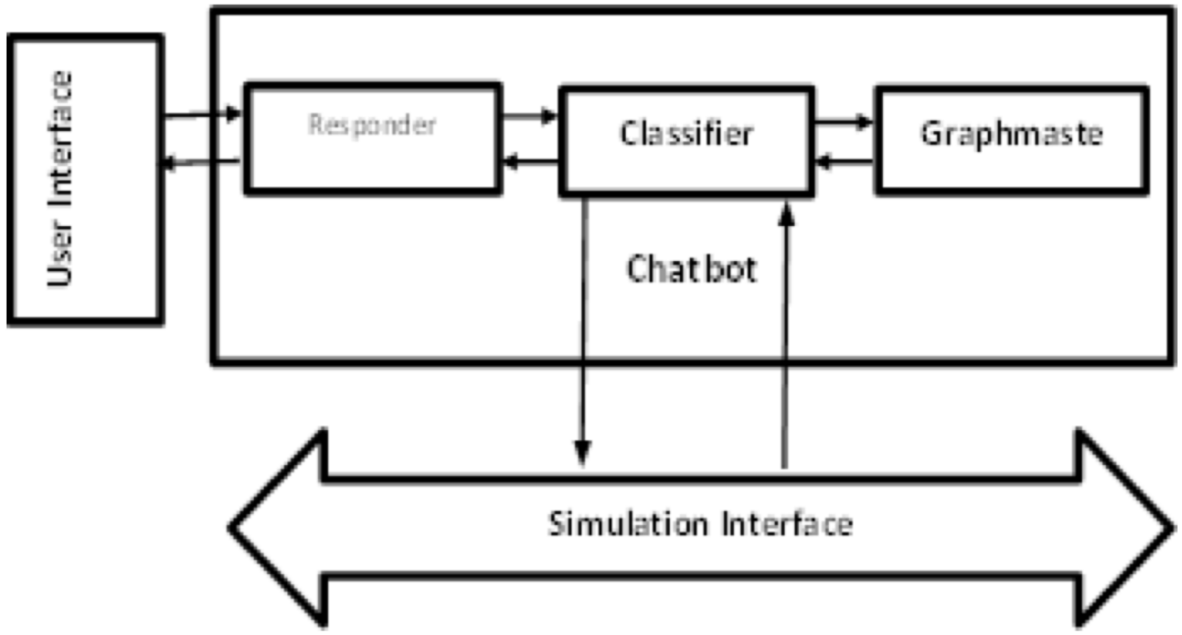


Figura 2.3: Componentes de um *chatbot*.

No âmbito de um *chatbot*, o Processamento de Linguagens Naturais - que aborda a geração e compreensão automática de línguas humanas naturais - inclui 5 passos importantes[6]. A **análise léxica** corresponde à identificação e análise da estrutura das palavras e divide o texto por capítulos, frases e palavras. A **análise sintática** envolve a análise da gramática e organização das palavras, de modo a que as relações entre as palavras se tornem claras e façam sentido. A **análise semântica** verifica o texto para ver se tem significado e desenha esse exato significado mapeando construções sintáticas. **Integração do discurso** e **análise pragmática** trabalham na interpretação final da mensagem real do texto, já que o significado de qualquer frase depende do seu contexto.

Os sistemas de pergunta-resposta (*question answering*) têm fortes contornos de Processamento de Linguagens Naturais e podem ser divididos em duas categorias: domínio fechado[7] e domínio aberto[8]. O primeiro abrange apenas diálogo sob um específico tema, como compras de roupa ou meteorologia, e são naturalmente mais fáceis de codificar, obtendo também melhores resultados. O segundo concentra-se num sistema mais abrangente a outros temas, tendo de ser capaz de responder a perguntas que não estejam limitadas apenas ao campo específico.

Os *bots* bancários estão, mais ou menos, no meio destas duas categorias. Por um lado não se podem comportar exclusivamente como um domínio fechado, pois têm de oferecer uma maior interação do que aquela que é dada pelas ATMs, em que são apenas selecionadas as escolhas, e permitir, de certa forma, ao utilizador que este se possa sentir confortável para prolongar o diálogo até um nível de confiança em que fique esclarecido e seguro com os *inputs* que está a receber. Por outro lado, os *chatbots* não podem ser tão díspares com possíveis diálogos, como outros que simulam um certo nível de conversa que se assemelha a um psicólogo[9]. Isto é, se

queremos um *bot* que consiga falar normalmente e que não mostre desconhecimento para outras falas que receba que não sejam exclusivas da sua área de incisão, domínio fechado não funciona; se querermos tornar o *bot* eficiente e conduzir o utilizador para o seu âmbito de funcionamento, este também não se deve comportar inteiramente como um domínio aberto.

Além disto, os *bots* bancários representam ainda outra categoria dos sistemas de diálogo, que é o facto de serem orientado para tarefas. Assim, se um utilizador quer abrir uma conta e fornece os seus dados, o *chatbot* tem de estar associado a uma base de dados em que lhe seja possível fazer o *insert* dos dados do utilizador. Existem outras operações, nomeadamente a transferência de dinheiro, que já envolvem duas contas no banco e, por conseguinte, duas operações. E, em geral, os sistemas de diálogo orientados para tarefas fornecem uma estrutura para que quem utilize o *bot* tenha conversas com a máquina para concluir uma tarefa específica. Normalmente os *chatbots* são sistemas não orientados a tarefas, não tendo a intenção de ajudar o utilizador a concluir qualquer tarefa específica e só se concentrando nos diálogos, o que já é difícil de si[10], e por isso os *bots* bancários são ainda mais ardilosos de desenvolver.

Por todas estas questões, ainda não existe grande ênfase da comunidade científica dos *chatbots* na área bancária. Outra das condicionantes é a proteção e privacidade dos dados das pessoas, o que faz com que apenas os bancos possam ser autores de *bots*, uma vez que *developers* que queiram construir os seus próprios *chatbots* para realizar serviços bancários, tenham de os fazer sob um ambiente virtual, como a criação de bases de dados, com representantes e contas fictícias.

2.1.2.1 Interfaces de conversação

Bots da área bancária podem funcionar apenas como *assistant systems*, que direcionam os utilizadores para outros serviços que - esses sim -, resolverão o problema[11]. Estes agentes de *software* podem facilmente tratar casos mais gerais do que *chatbots*, que resolvem tudo no seu sistema embebido, já que fornecem uma plataforma para integrar funções de terceiros no assistente. A Siri, da Apple, apesar de não estar relacionada com o tema bancário, é um bom exemplo de um *bot* deste tipo.

Com o aparecimento de plataformas de conversação, como o Messenger ou o Skype, e a capacidade de integrar componentes de *software* de terceiros por meio de APIs, vários *bots* bancários aumentaram a sua interação com os seus utilizadores, através dessas mesmas plataformas[12].

O *chatbot* tem de ser capaz de agir de várias maneiras consoante o *input* de diálogo que lhe é passado pelo utilizador. O *bot* responde a uma frase de uma linguagem natural, como o português, e tenta fazer uma ação com base na fala dada pelo humano[13]. Durante uma conversa, o *chatbot* procura o contexto (intenção) para realizar a ação pretendida[14], mas se essa mesma intenção aparecer numa próxima entrada, este tem de ser capaz de utilizar a informação que já foi dada anteriormente, sem ter de solicitar os dados novamente ao utilizador por via de uma pergunta. Por exemplo, para o caso de um *bot* de meteorologia, o *bot* não vai perguntar em

todas as instâncias qual é a localização do utilizador para quando este solicitar ver novamente o tempo[15]. Vejamos a figura abaixo:

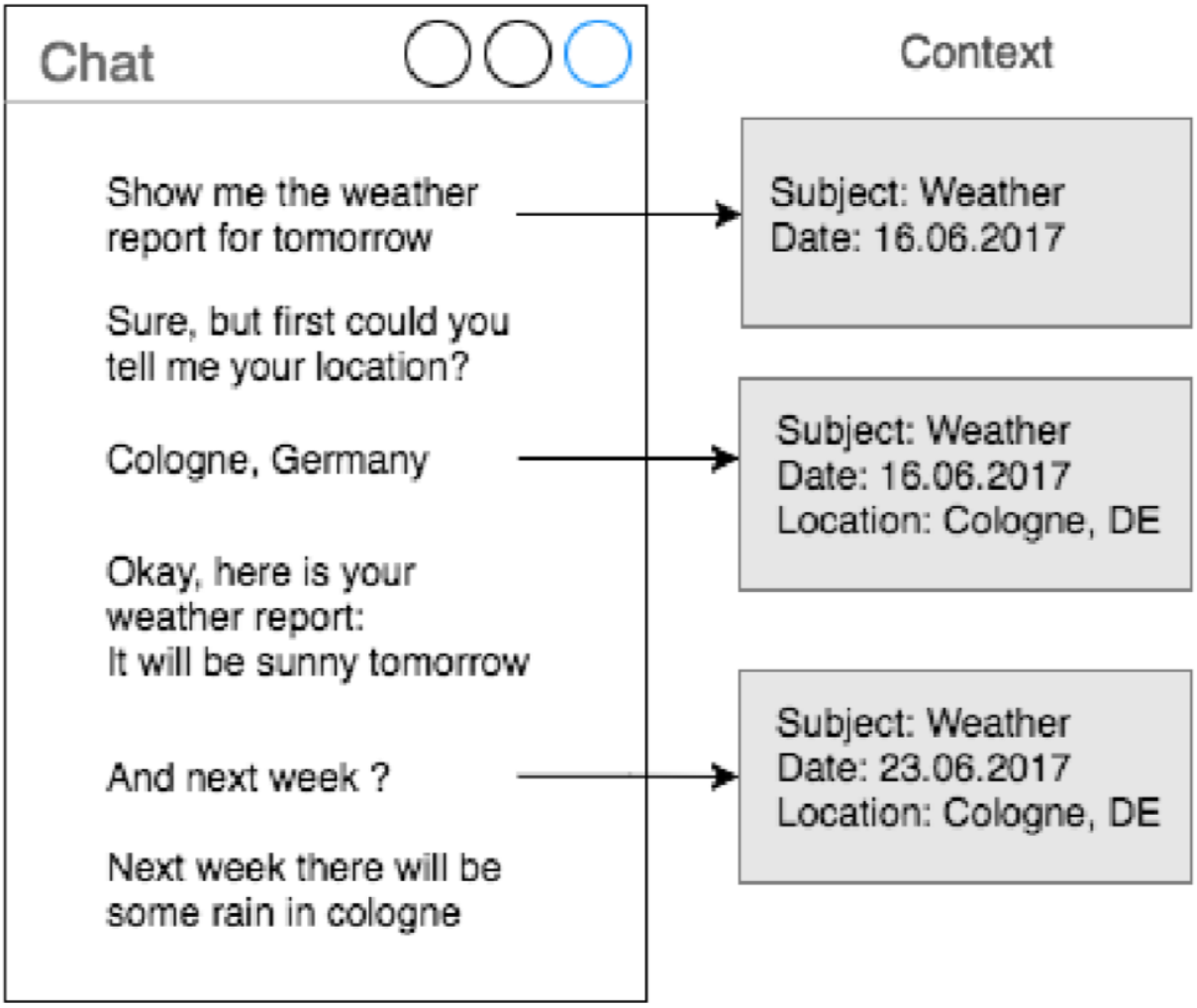


Figura 2.4: Representação esquemática do processo de extração do contexto num *chatbot*[1].

Podemos ver que o funcionamento do *bot* representado necessitava do preenchimento de 3 *slots*: o tema, a data e a localização. No entanto, é importante saber quando é necessário ao agente manter estes valores e usá-los em ações futuras. No caso bancário é igual. Imaginemos o seguinte caso:

Transfere-me 40 euros para o Paulo

Entidade: TransferirDinheiro; **Montante:** 40 euros; **Destinatário:** Paulo.

E também para a Maria

Entidade: ?; **Montante:** ?; **Destinatário:** Maria.

O *bot* tem ser perspicaz e perceber que a entidade e o montante a utilizar são os mesmos da ação anterior. Exemplos mais simples podem ocorrer aquando do momento em que o utilizador

diz o seu nome e esse valor tem de ficar não só armazenado para toda a sessão, mas para cada vez que o utilizador inicia uma interação com o *chatbot*.

2.1.2.2 Casos de utilização

Os principais bancos de Wall Street já aderiram ao fenómeno, enraizando diferentes *chatbots*, com diferentes funções, nos seus serviços. Alguns bancos usam o *bot* para enviar notificações aos seus clientes, outros para facilitar as operações bancárias, outros ainda para auxiliar em poupanças, assim como para consultar os registos. E em casos particulares, a tecnologia até é utilizada para resolver processos antigos, poupando dezenas de horas em trabalho.

Podemos ver o que tem sido feito na área bancária a nível profissional quando analisamos o contexto que os grandes bancos a nível mundial estão a dar aos seus *chatbots*[16]:

1. Bank of America
    - O Bank of America (líder de mercado na utilização de serviços bancários móveis nos Estados Unidos) introduziu, dentro da aplicação móvel do banco, o *bot* Erica (da palavra AmErica) para enviar notificações aos consumidores, fornecer informações sobre o saldo, sugerir métodos de poupança, oferecer atualizações de relatórios de crédito, pagar contas e ajudar os clientes com transações simples.
2. JPMorgan Chase
    - A JPMorgan Chase está a utilizar *bots* para otimizar as suas operações de *back-office*. O banco lançou o COIN para analisar contratos complexos de forma mais rápida e mais proficiente do que advogados. Segundo o banco, a introdução do *bot* economizou mais de 360.000 horas de trabalho.
3. Wells Fargo
    - O *chatbot* da Wells Fargo combina Inteligência Artificial com o Facebook Messenger para responder às mensagens em linguagem natural dos utilizadores, como o saldo disponível nas contas e a localização da caixa de ATM do banco mais próxima.
4. Capital One
    - A Capital One introduziu um assistente de conversação baseado em texto chamado Eno para ajudar os clientes a gerir o dinheiro usando os seus telemóveis. Os clientes podem obter informações do *chatbot* sobre o saldo da conta ou histórico de transações e pagar as contas instantaneamente.
5. Ally Bank
    - O Ally Assist é um assistente virtual dentro da aplicação móvel do Ally Bank e pode ser acedido via voz ou texto para executar funções como efetuar pagamentos,

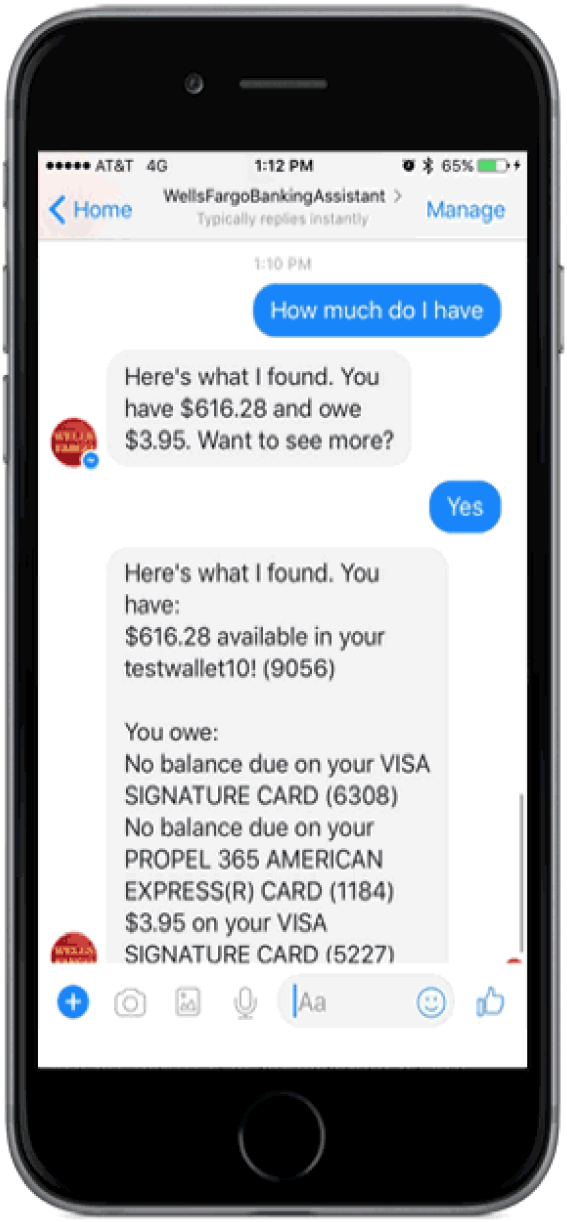


Figura 2.5: WellsFargoBankingAssistant.

transferências, transações P2P e depósitos. Um cliente também pode solicitar um resumo de conta ou o histórico de movimentos, bem como monitorizar os padrões de economia e gastos.

6. American Express

   - A American Express oferece benefícios para os seus clientes com a ajuda dos *chatbots*, que possibilitam notificações de venda em tempo real, recomendações contextuais e lembretes sobre benefícios do cartão de crédito.

7. HSBC

   - O *bot* Amy é uma plataforma de atendimento ao cliente que assume a forma de um

Virtual Assistant Chatbot para serviços bancários corporativos no HSBC de Hong Kong. A Amy pode oferecer um suporte instantâneo a consultas de clientes numa base 24 por 7 e está disponível em computadores e dispositivos móveis em inglês e chinês tradicional e simplificado.

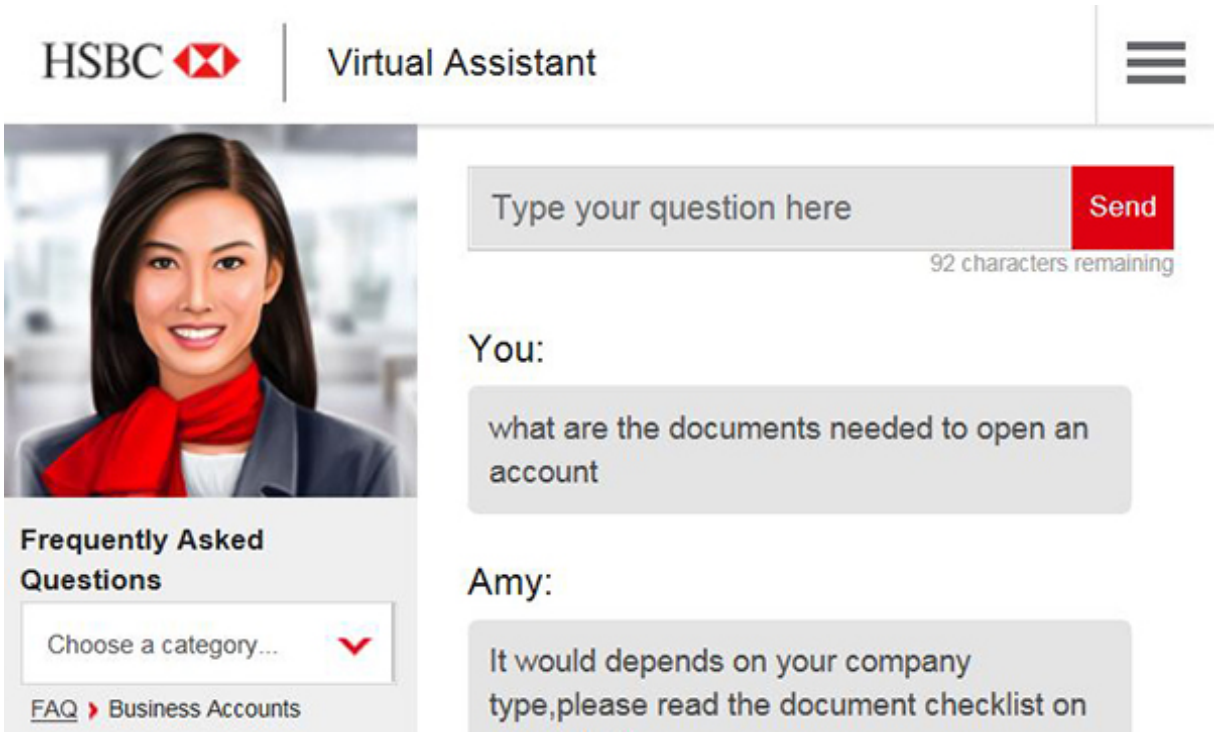


Figura 2.6: HSBC Virtual Assistant.

8. Hang Seng Bank

   - O Hang Seng Bank introduziu dois *chatbots*. O HARO, disponível *online* e por meio da *app* do banco, lida com consultas gerais sobre produtos e serviços do banco, como o cálculo de pagamentos de hipotecas, enquanto que a DORI é responsável por encontrar descontos em restaurantes e fazer recomendações com base nas preferências dos clientes. A DORI está incorporada no Facebook Messenger e funciona de forma semelhante à Siri, da Apple.

9. Commonwealth Bank

   - O Commonwealth Bank lançou um *chatbot* chamado Ceba para ajudar os clientes e conta com mais de 200 tarefas bancárias, como ativar o cartão, verificar o saldo da conta, fazer pagamentos ou levantar dinheiro.

10. Santander UK

    - O Santander do Reino Unido disponibilizou um *bot* de reconhecimento de voz que permite aos seus utilizadores fazerem pagamentos, saberem o saldo, declararem perdas de cartões, configurarem alertas de conta, entre outros.

11. Barclays Africa (ABSA)

    - O ABSA lançou o ChatBanking para atender os consumidores onde quer que eles estejam, já que está presente nas redes sociais e aplicações móveis mais populares do Mundo, como o Twitter ou o Messenger.

12. DBS Singapore

    - O Digibank da DBS oferece aos utilizadores um assistente virtual 24/7 na aplicação do banco. Os clientes podem interagir com o assistente via voz ou texto, sendo que este consegue responder a mais de 10.000 perguntas comuns relacionadas com assuntos bancários.

13. Royal Bank of Scotland

    - O RBS está a usar um *chatbot* chamado Luvo para automatizar e agilizar o seu serviço *online* para o cliente. Desenvolvido com o auxílio da plataforma de inteligência artificial da IBM, o Watson, o Luvo lida com perguntas simples, libertando a equipa de suporte para se concentrar nas questões mais difíceis.

Em Portugal ainda nenhum banco ou instituição financeira adotou a tecnologia, não aproveitando desta forma os benefícios que os *chatbots* podem trazer para as suas operações.

2.2 Ferramentas para o desenvolvimento de *bots*

2.2.1 Plataformas de desenvolvimento rápido

Plataformas como o Chatfuel, o Botsify, etc., ajudam na criação de *chatbots* sem requererem grandes aptidões de programação. Permitem que se adicione funcionalidades ao *bot*, como integração para outras APIs (Facebook Messenger, por exemplo), fluxo de conversação ou condições de aprendizagem.

Estas plataformas não se concentram tanto nos diálogos, mas mais nas respostas através de escolhas por intermédio de *clicks*. Assim, apesar de serem de fácil utilização e de simples entendimento, não são apelativas para quem pretende atingir um certo nível de detalhe e sofisticação.

2.2.2 *Frameworks* para programação

Bem diferente do exemplo das plataformas para o desenvolvimento de *bots*, são as *frameworks*. Estes serviços só têm utilidade para programadores com *skills* em receber e tratar dados (JSON/AIML/XML) com código. Assim, as plataformas permitem a criação de *intents* e *entities*, que mais tarde servirão para reconhecimento nos diálogos. Um *intent* representa o propósito

Figura 2.7: Os 5 maiores bancos dos EUA que adotaram *chatbots*. Retirado de Maruti Techlabs.

do *input* do utilizador e é definindo para cada tipo de solicitação. Já as *entities* representam o objeto que é importante para o *intent* e têm valores associados.

Exemplo de *input* do utilizador para um *bot* bancário: "Quero transferir 100 euros

para o Pedro";

Exemplo de um *intent*: TransferirDinheiro;

Exemplo de *entities*: Montante; Destinatário;

Valores das *entities*: 100 euros; Pedro.

As melhores *frameworks* para servirem como auxiliador na criação de um *bot*, e todas adquiridas por empresas multimilionárias, são:

- LUIS (Microsoft)
- Wit.ai (Facebook)
- Api.ai (Google)
- Watson (IBM)

2.2.3 Biblioteca `Chatterbot` do Python

Uma das ferramentas auxiliares na construção de *chatbots* é a biblioteca `Chatterbot` do Python, que torna mais fácil a geração automática de respostas quando é dado um *input* de utilizador. Consiste num mecanismo de diálogo conversacional baseado em *Machine Learning* e construído em Python, que torna possível gerar respostas de conversas conhecidas. O *design* independente da linguagem do `ChatterBot` permite que este possa ser treinado para falar qualquer idioma.

Uma instância não treinada do `Chatterbot` começa sem conhecimento de como comunicar. Cada vez que o utilizador redige uma frase, a biblioteca guarda não só essa resposta como também o texto à qual a frase foi respondida. À medida que o `Chatterbot` recebe mais *inputs*, a precisão das suas respostas aumenta. O processo de escolha da resposta funciona através da procura da instrução conhecida mais parecida com o *input*, retornando depois a resposta mais provável a essa frase, através da frequência com que cada resposta é emitida pelos utilizadores com que o *bot* comunica.

A libraria `Chatterbot` reúne contornos de aprendizagem, pois está sempre a adquirir conhecimento quando comunica e, quanto mais treinada estiver, mais próximos do tema são os seus diálogos, chegando mesmo o *bot* a emitir falas que não estavam pré-preparadas. É por isso demais visível que o `Chatterbot` é mais do que um simples gerador de respostas previamente fornecidas e é também capaz de se comportar como próprio criador de conversas.

O `Chatterbot` recebe contribuições de utilizadores colaboradores para os módulos dos *datasets* de diferentes línguas que tem presentes no GitHub da biblioteca e podem ser usados para treinar as falas dos *bots*. Este arquivo[3] contava inicialmente com três línguas - inglês, espanhol e português -, mas neste momento já são quase 20 os idiomas englobados. Apesar de tudo, é de

[3]https://github.com/gunthercox/chatterbot-corpus/tree/master/chatterbot_corpus/data

realçar que o português existente na lista é o do Brasil, mas que será feito um esforço para que todo o conteúdo gerado neste projeto a nível de diálogos possa ser partilhado uma vez findo o trabalho e alcançadas as metas com sucesso.

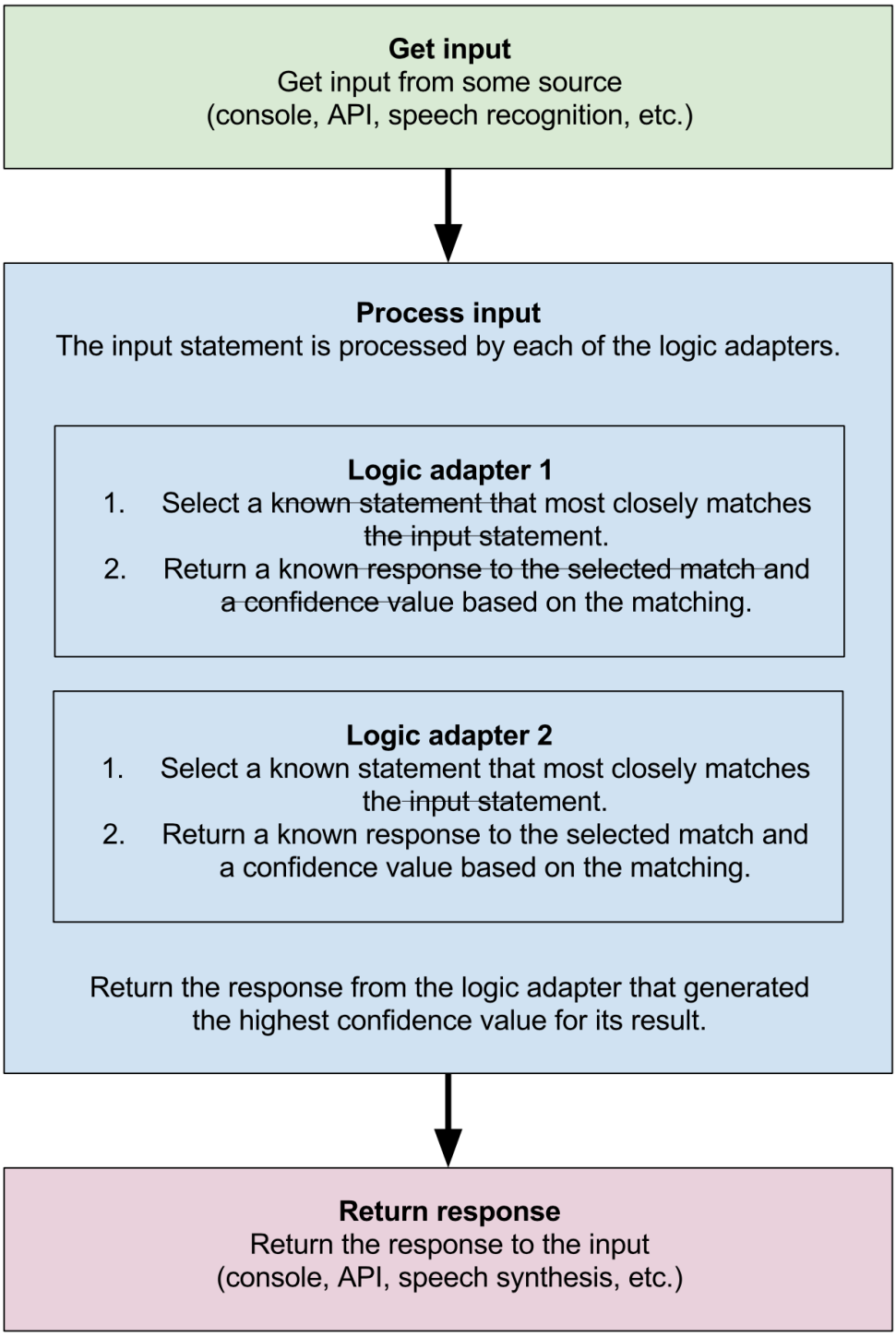


Figura 2.8: *Process Flow Diagram* da biblioteca do `Chatterbot`.

2.2.4 NLTK

O NLTK[17], *Natural Language Toolkit*, é uma plataforma do Python para trabalhar com dados de linguagem humana. Fornece ao utilizador uma série de serviços para tratar as frases e palavras, como *Stemmers*, que removem afixos morfológicos das frases, ficando apenas com o radical, ou *Tokenizers*, que dividem as palavras de uma respetiva frase ou até que separam as frases de um texto.

Este conjunto de bibliotecas - mais de 50 -, permite classificação, tokenização, *stemming*, *tagging*, análise semântica e possibilita a introdução de Processamento de Linguagens Naturais num programa[18].

O NLTK foi desenvolvido pelos professores Steven Bird e Edward Loper do Departamento de Ciência de Computadores e Informática da Universidade da Pensilvânia em 2001, e continua a ser hoje um utensílio de ensino para os estudantes da área e uma ferramenta muito útil na construção de sistemas de Inteligência Artificial[19].

```
1  from nltk.tokenize import sent_tokenize, word_tokenize
2
3  ex = "Olá Sr. António, o meu nome é Fábio. Tenho 22 anos. Moro em Portugal."
4
5  print("\nFRASES:")
6  for i in sent_tokenize(ex):
7      print(i)
8
9  print("\nPALAVRAS:")
10 for i in word_tokenize(ex):
11     print(i)
12
13 print("\n")
```

```
Descargas — -bash — 88×29
fabioteixeira2up:Downloads fabioteixeira$ python3 bot.py

FRASES:
Olá Sr. António, o meu nome é Fábio.
Tenho 22 anos.
Moro em Portugal.

PALAVRAS:
Olá
Sr.
António
,
o
meu
nome
é
Fábio
.
Tenho
22
anos
.
Moro
em
Portugal
.

fabioteixeira2up:Downloads fabioteixeira$
```

Figura 2.9: Exemplo de utilização dos *Tokenizers* no NLTK.

2.2.5 Linguagens para representação de informação

2.2.5.1 JSON

JSON[20] (*JavaScript Object Notation*) é um formato de ficheiros *open source* que usa texto legível para transmitir objetos de dados baseados em pares do tipo atributo-valor. É frequentemente utilizado em comunicações assíncronas *browser*-servidor, substituindo o XML[21]. O formato de dados, que deriva do JavaScript, permite o armazenamento de informações de maneira organizada e de fácil acesso. Oferece um conjunto de dados legível por humanos, possível de aceder de uma forma realmente lógica[22].

As duas principais partes constituintes do JSON são os atributos/chaves e os valores. Juntos, formam um par chave-valor.

- Chave: uma chave é sempre uma cadeia entre aspas;
- Valor: um valor pode ser uma *string*, um número, uma expressão booleana, uma matriz ou um objeto;
- Par chave-valor: um par de chaves ou atributos e valores segue uma sintaxe específica, com a chave a ser seguida por dois-pontos, e posteriormente pelo valor. Os pares chave-valor são separados por vírgulas.

```
1  var family = [{
2      "name" : "Jason",
3      "age" : "24",
4      "gender" : "male"
5  },
6  {
7      "name" : "Kyle",
8      "age" : "21",
9      "gender" : "male"
10 }];
```

gistfile1.js hosted with ♥ by GitHub view raw

Figura 2.10: Exemplo de um ficheiro JSON.

2.2.5.2 AIML

AIML[23] (*Artificial Intelligence Markup Language*) é uma linguagem, baseada em XML[24], que permite definir interações únicas, compostas por um estímulo produzido pelo utilizador e uma resposta correspondente. Cada interação é definida através da estrutura de categorias do

sistema. Uma categoria contém dois elementos: o padrão e o modelo. O padrão descreve um possível estímulo do utilizador. O modelo caracteriza a resposta correspondente do sistema. Às vezes, o modelo é contido pelo elemento. Este elemento permite gerir o contexto numa conversa. Isto é, um intérprete AIML pode usar diretamente o conteúdo do modelo como resposta. Outras vezes, o conteúdo de um modelo refere-se a outras categorias. Neste caso, a resposta do sistema é obtida combinando o conteúdo de muitas categorias.

Quando o utilizador fornece um *input*, o interpretador AIML tenta associar esse *input* com um padrão. A correspondência pode ser executada em várias etapas. O primeiro elemento encontrado é a categoria. O segundo elemento correspondente é o padrão derivado das categorias especiais permitidas na língua, se presentes. Finalmente, é testada ainda uma correspondência com o atributo *name* no elemento *topic*, permitindo assim distinguir o tópico de uma conversa. O mais famoso *chatbot* programável com código AIML é o A.L.I.C.E.[25], *bot* de 1950 baseado no Teste de Turing.

```
<aiml:aiml version 1.01 xmlns:aiml = "http://alicebot.org/2001/AIML">
<aiml:topic name =".."> </topic>
<aiml category>
    <aiml pattern> ..
    </aiml pattern>
    <aiml template>..
    </aiml template>
</aiml category>
</aiml>
```

Figura 2.11: O *chatbot* A.L.I.C.E., programado em 1950, através da linguagem AIML.

O estudo e levantamento das ferramentas abordadas neste capítulo foi importante para a definição da abordagem a ser seguida no desenvolvimento.

Capítulo 3

Análise de Requisitos

Este capítulo tem como propósito a análise de requerimentos e necessidades estabelecidas para a estrutura da solução. Concretamente, o planeamento sobre o comportamento que o *bot* deveria adquirir no momento da sua implementação.

3.1 Público alvo

O facto de o *chatbot* ter como índole um *software* que trata de funções da área da banca, como a criação de contas, atualização de saldo, transferências, etc., faz com que o público alvo da tecnologia sejam pessoas de uma faixa etária compreendida entre os 18 e os 80 anos. Os serviços realizados incluem dados importantes como os montantes envolvidos nas operações e nesse sentido, o lógico é que o consumidor que vai utilizar a aplicação esteja inserido numa idade adulta e com plena consciência das suas ações.

3.2 Problemas do consumidor

Por trás de uma solução tem de haver sempre um problema e no caso dos *chatbots* a temática não é diferente. Os utilizadores vivenciam algumas dificuldades, que podem ser corrigidas, com a aplicação de um *bot* bancário. Exemplos de problemas:

- Necessidade de ser o utilizador a ter de realizar as tarefas;
- Impossibilidade de contactar ajuda especializada sempre que o consumidor deseja;
- Falta de capacidade, especialmente no caso de pessoas mais idosas, em confiar num sistema baseado apenas em operações e botões;
- Pouca flexibilidade e eficiência na recolha de informações;
- Desperdício de tempo e distância para realizar os serviços.

3.3 Problemas das empresas

Também as empresas estão a aderir aos *chatbots*, nomeadamente para resolver problemas que enfrentam e para melhorar o rendimento dos negócios. Exemplos de problemas:

- Necessidade de atendimento físico e demorado ao consumidor quando este o requer;
- Obrigatoriedade de ter trabalhadores em funções que podem perfeitamente ser realizadas por tecnologia;
- Gastos desnecessários em serviço de suporte;
- Dificuldades na interação com os clientes, que esperam muito tempo no atendimento para verem resolvidas as suas questões;
- Pouca eficiência no plano de negócios.

3.4 Resolução de problemas

Os *chatbots* resolvem estes problemas, associados aos consumidores e às empresas, e melhorar a relação entre ambos. No caso dos utilizadores, estes recebem um atendimento instantâneo, mas que ao mesmo tempo se assemelha com o carácter humano, pois representa uma interface similar ao de mensagens que praticamente todas as pessoas usam. Também as empresas ganham pela evolução da tecnologia, por se aproximarem do consumidor, pelo que podem poupar financeiramente e pelo *boost* financeiro que a introdução dos *bots* pode fazer aos negócios. Os trabalhadores não precisam de fazer simples tarefas que passam a ser realizadas pelo *chatbot*, e a rapidez na interação com o utilizador faz com que haja uma melhoria não só na qualidade do atendimento como também na quantidade de operações e tarefas realizadas por dia, mês ou ano, o que aumenta de forma visível a eficiência nos negócios das empresas.

3.5 Requisitos funcionais

Uma vez levantados os problemas e revista a forma como os *chatbots* podem, de facto, resolver essas divergências, listaram-se os requisitos, seja ao nível de processos, dados ou restrições operacionais, relacionados com aquilo que o sistema deve fazer. A saber:

- Ter a capacidade de se conseguir associar com o utilizador através de um diálogo simples e que inclui temáticas básicas, como agradecimentos, despedidas ou mensagens de boas-vindas;
- Permitir ao utilizador conseguir criar uma conta, de tipo corrente ou poupança, de acordo com a sua escolha;

- Permitir ao utilizador aceder ao seu saldo a qualquer momento da conversa, desde que forneça o seu nome;
- Permitir ao utilizador depositar dinheiro na sua conta;
- Permitir ao utilizador levantar dinheiro da sua conta;
- Permitir ao utilizador transferir dinheiro para outro utilizador, desde que este exista;
- Ter uma base de dados para manter todos os dados dos utilizadores, simulando as contas dos mesmos.

3.6 Requisitos não funcionais

Relativamente à qualidade global do *software*, destacam-se:

- Capacidade para a qualquer momento desenvolver as conversações, em especial adicionar mais diálogos de *small talk*;
- Facilidade na utilização do *bot*, oferecendo uma mensagem de boas-vindas bastante intuitiva e listando as suas funcionalidades para o utilizador;
- Exclusão de qualquer abstração, isto é, mesmo quando o *chatbot* não percebe a intenção da frase do utilizador, dizer que não percebeu e não entrar em aleatoriedade;
- Facilidade de manutenção, pela necessidade apenas de uma subscrição no Azure, que posteriormente liga o *bot framework* LUIS ao código;
- Rapidez no desempenho do *software* (as respostas são praticamente instantâneas);
- Facilidade na acessibilidade ao *bot*, sendo este bastante *friendly-user* na sua interação.

3.7 Testes preliminares

Ao longo do trabalho, foram seguidas três abordagens diferentes para a realização do mesmo, sendo que as duas primeiras acabaram por revelar-se insuficientes para o pretendido.

3.7.1 `Chatterbot`

A primeira abordagem seguida para o desenvolvimento do *bot* foi a utilização da biblioteca `Chatterbot` do Python, em conjunto com um *dataset* de diálogos.

No código-fonte, numa primeira instância, importou-se o *chatbot* e o método de treino, depois criou-se o *chatbot* e definiu-se o *trainer*.

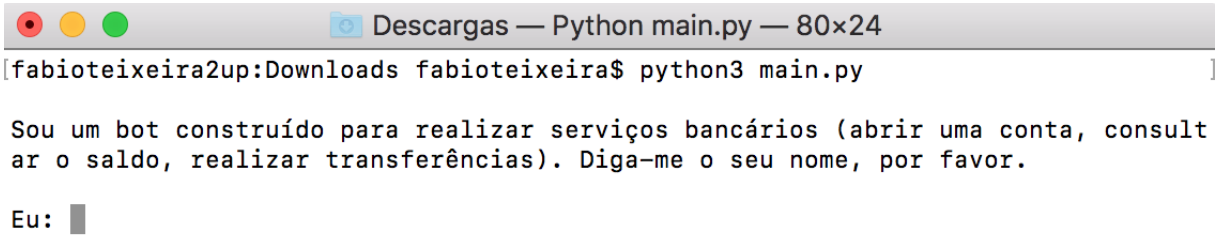


Figura 3.1: Mensagem de boas-vindas do *chatbot* proposto.

```
$ brew install python3
$ sudo pip3 install chatterbot
```

Bloco de Código 3.1: Requerimentos para utilizar a biblioteca `Chatterbot`

De seguida foi preciso ler os diálogos previamente criados. Para isso, criaram-se diversos ficheiros, para diferentes tipos de conversações. Um dos exemplos retrata os diálogos de saudação, que foram colocados num ficheiro chamado *hello.txt*. Para além das saudações, criou-se um ficheiro para término de conversa (*bye.txt*), um para respostas de sim/não/talvez (*confirmation.txt*), um para reações emotivas e de riso (*emotion.txt*), um para elogios (*courtesy.txt*), um para informação sobre o *bot* (*agent.txt*), um para informação sobre o utilizador (*user.txt*), um para outras respostas (*other.txt*) e vários para adereçar o problema bancário.

Os ficheiros foram então colocados numa pasta chamada *arq*, sendo que depois foi utilizado um ciclo `for` a correr toda a pasta lendo os arquivos e treinando o *bot* com esses ficheiros. De notar que nem sempre é preciso treinar cada vez que o programa é compilado, mas para isso é necessário acrescentar uma condição de `read_only=True` ao *chatbot*.

Para finalizar, para construir o diálogo, utilizou-se um *input* pedido ao utilizador como *quest* e uma resposta do *bot* gerada pelo atributo `get_response`. Ainda, se a taxa de confiança fosse inferior a 0.5 era dito apenas *"Desculpe, não percebi."*.

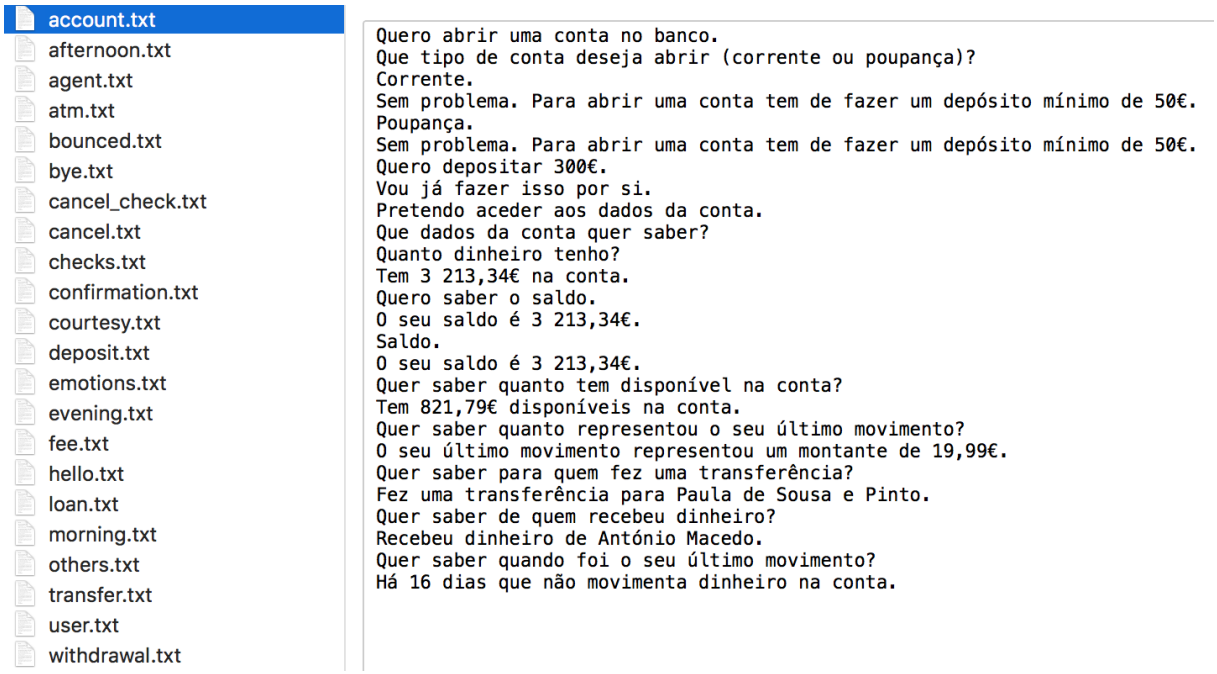


Figura 3.2: Ficheiros do *dataset* de diálogos.

```
from chatterbot.trainers import ListTrainer // metodo para treinar o chatbot
from chatterbot import ChatBot // importar o chatbot
import os

bot = ChatBot('Teste')
bot.set_trainer(ListTrainer) // definir o "treinador"

for arq in os.listdir('arq'): // procurar na pasta dos arquivos
   chats = open('arq/' + arq, 'r').readlines() // ler os ficheiros
   bot.train(chats) // treinar os ficheiros

while True:
   resq = input('Eu: ')
   resp = bot.get_response(resq)

   if float (resp.confidence) > 0.5:
      print('Bot: ' + str(resp))
   else:
      print('Bot: Desculpe, nao entendi.')
```

Bloco de Código 3.2: Código do *bot* utilizando a biblioteca `Chatterbot`.

Apesar de se ter atingido algum nível de sofisticação, esta não foi a abordagem seguida até final, até porque tem contornos de NLU (*Natural Language Understanding*), ao invés de NLP. Isto é, o programa tratava apenas dos casos estudados e para cada frase tinha uma resposta programada. No entanto, quando não encontrava uma frase previsível e com uma resposta

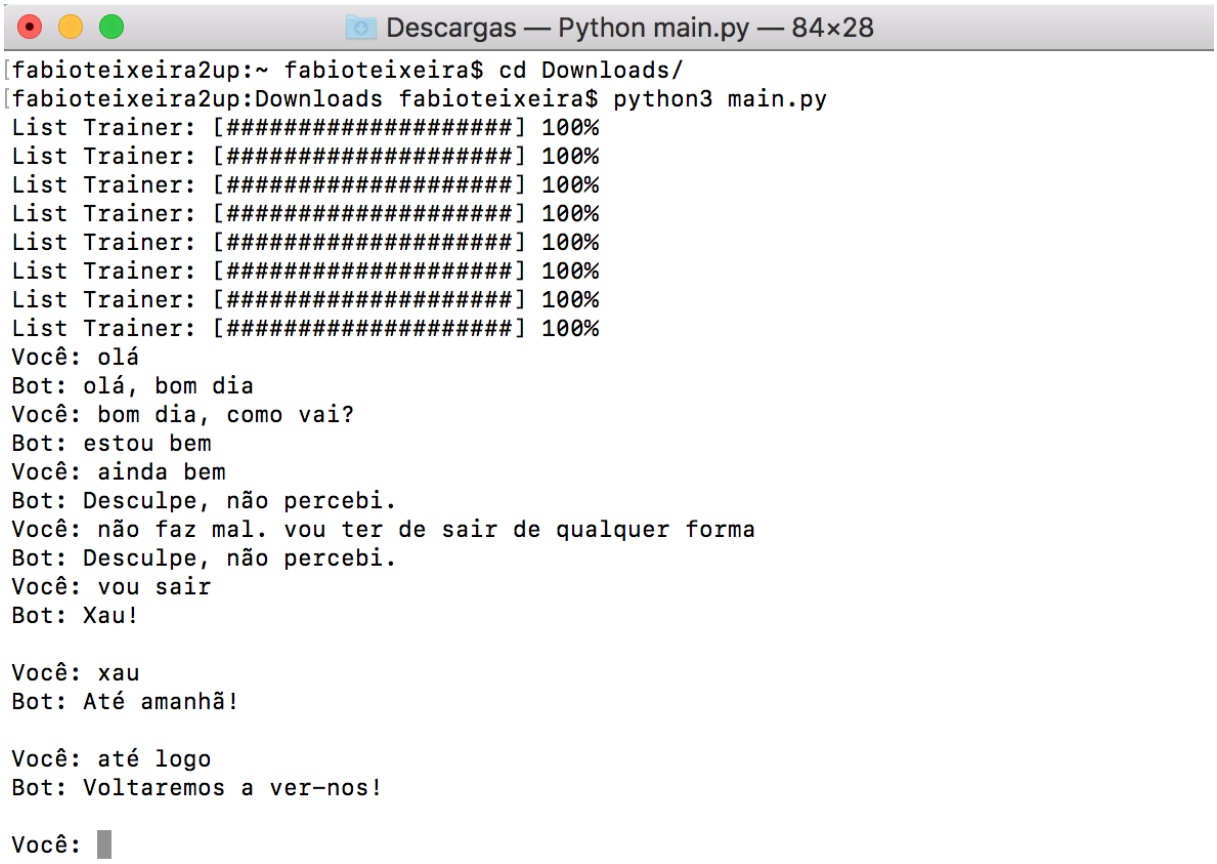


Figura 3.3: Um exemplo de conversação e interação com o *bot*. *Print screen* tirado a 16/01/2018.

associada, não era capaz de oferecer uma resposta "inteligente". Ao contrário de NLP, *Natural Language Processing*, NLU não tem um classificador para definir a intenção da fala e dessa forma não é capaz de ter a habilidade de perceber significados e de determinar, com mais aproximação e exatidão, o que o utilizador pretende e responder-lhe da mesma forma[26].

3.7.2 Criação de *intents* + NLTK

Para satisfazer as necessidades do programa em se comportar como NLP, foi seguida a abordagem de se criar, manualmente, *intents* e classificá-los através de ferramentas do NLTK.

Posteriormente, estes dados em JSON foram tratados para classificar o *input* do utilizador e perceber a sua intenção (*intent*). Para isso foram criadas várias listas, com as frases associadas à *tag* (*intent*), frases associadas às respostas, e respostas, também com a *tag*.

Com o *input* do utilizador, utilizou-se o *word_tokenize* para o dividir em palavras e tentar encontrar a *tag* numa primeira instância (nível de importância), e retornar a resposta associada a essa *tag*. Se tal não fosse possível, procurar-se-ia então na lista de frases que estavam guardadas, com a *tag* associada, e posteriormente também se retornaria a resposta. E, se, em todo o caso, não fosse encontrada nenhuma palavra do *input* do utilizador em nenhuma *tag* ou frase (*patterns*

```
{"intents":
    [
        {
            "tag": "ola",
            "patterns": ["oi", "ola", "boas", "esta alguem aqui", "bom dia",
                "boa tarde", "boa noite"],
            "responses": ["Ola, seja bem vindo!", "Ola, como posso ajuda-lo?",
                "Ola,    bom ve-lo de novo."]
        },
        {
            "tag": "adeus",
            "patterns": ["adeus", "vou sair", "xau", "ate amanha", "ate logo"],
            "responses": ["Estarei aqui quando voltar.", "Eu fico por aqui!",
                "Voltaremos a ver-nos."]
        },
        {
            "tag": "obrigado",
            "patterns": ["obrigado", "muito obrigado", "obrigado pela ajuda"],
            "responses": ["De nada!", "O prazer e meu."]
        },
        {
            "tag": "conta",
            "patterns": ["quero abrir uma conta", "abre-me uma conta", "quero
                criar uma conta", "cria-me uma conta", "criar nova conta",
                "abrir conta"],
            "responses": ["Que tipo de conta deseja abrir (corrente ou
                poupanca)?"]
        },
        {
            "tag": "transfere",
            "patterns": ["transfere 50 euros para o pedro", "envia 20 euros
                para a paula", "quero transferir 100 euros para o rui", "quero
                mandar 200 euros para a maria", "transfere 150 euros para o
                bruno", "transfere 120 euros para o rodrigo"],
            "responses": ["Tem a certeza que deseja fazer isso?"]
        }
    ]
}
```

Bloco de Código 3.3: Exemplos de *intents* utilizados com o NLTK.

do ficheiro JSON), era então dada a resposta de *"Desculpe, não percebi."*.

No entanto, mais uma vez, apesar de este método já se comportar como NLP, não era o mais satisfatório e, por conseguinte, não foi o utilizado até final. O processo de classificação de *intents* não era o melhor, pois era necessário que se detetasse a *tag* ou uma palavra das frases definidas para que o programa entendesse a intenção da fala. Desta forma também seria difícil lidar com a extração das entidades, até porque seria necessário listar as *entities* para cada *intent* e se a interação com o utilizador passasse para um plano que não estava definido seria muito difícil

(para não dizer impossível) garantir o bom funcionamento da classificação.

3.7.3 Comparação

Tabela 3.1: Comparação entre as abordagens.

Tarefa	`Chatterbot`	NLTK
Permite criação de diálogos	x	x
Tem método de treino	x	
Comporta-se como NLP		x
Descobre intenções		x
Descobre intenções por aproximidade		
Descobre entidades		
Descobre entidades com diferentes valores		
Descobre atributos dinamicamente		
É configurável	x	x
É configurável para outros temas		x
É gratuito	x	x

A análise de requisitos efetuada neste capítulo permitiu uma maior eficiência na implementação do *software*, uma vez que foi definido de forma detalhada aquilo que se pretendia que o *bot* fizesse. Também os testes preliminares, relativos às abordagens que não foram consumadas, valorizaram a solução proposta seguida, na medida em que esta cumpriu todas as condições de forma satisfatória.

Capítulo 4

Solução proposta

Neste capítulo é listado todo o processo de desenvolvimento, que consiste numa solução que utiliza um classificador de intenções do *bot framework* (LUIS) da Microsoft. Para a representação interna estruturada das mensagens em língua natural trocadas entre o humano e a máquina foi utilizada a notação JSON.

Para o processo de desenvolvimento, foi criado no LUIS um conjunto de intenções e entidades que mais tarde classifica as frases que o utilizador usa na conversa. A API transforma, depois, a informação em dados no formato JSON, que são recebidos na parte do código. Usando a linguagem de programação Python, são atribuídas diferentes respostas consoante a intenção das falas e para determinadas operações que o utilizador pretende efetuar, são realizadas ações com incidência na base de dados.

Face às abordagens testadas previamente, a forma encontrada para garantir a sofisticação pretendida no *chatbot* foi a utilização de uma API (LUIS) para classificar os *intents* e *entities* e mais tarde trabalhar com estes dados através do código em Python. Como vimos anteriormente, as *frameworks* para desenvolvimento de *bots* permitem a criação de intenções e entidades, bem como a adição de frases para programar o *bot*, para que este fique mais inteligente e comece a detetar por si estes atributos, sem que tenha de ser o utilizador humano a defini-los, um por um. Este método de classificação garante uma aproximação muito mais detalhada à intenção da fala do utilizador, que servirá para depois retornar respostas de forma mais precisa.

4.1 LUIS

O LUIS é um serviço baseado em *Machine Learning* para transformar linguagens naturais em aplicações, *bots* ou dispositovs IoT (*Internet of Things*). Oferece uma personalização facilitada para diversos produtos e é um parceiro muito requerido pelas empresas do meio[27].

Neste trabalho foi criada uma aplicação chamada ChatBanco, definindo-se *intents* e *entities* para utilizar na classificação das frases aprendidas pelo *bot*.

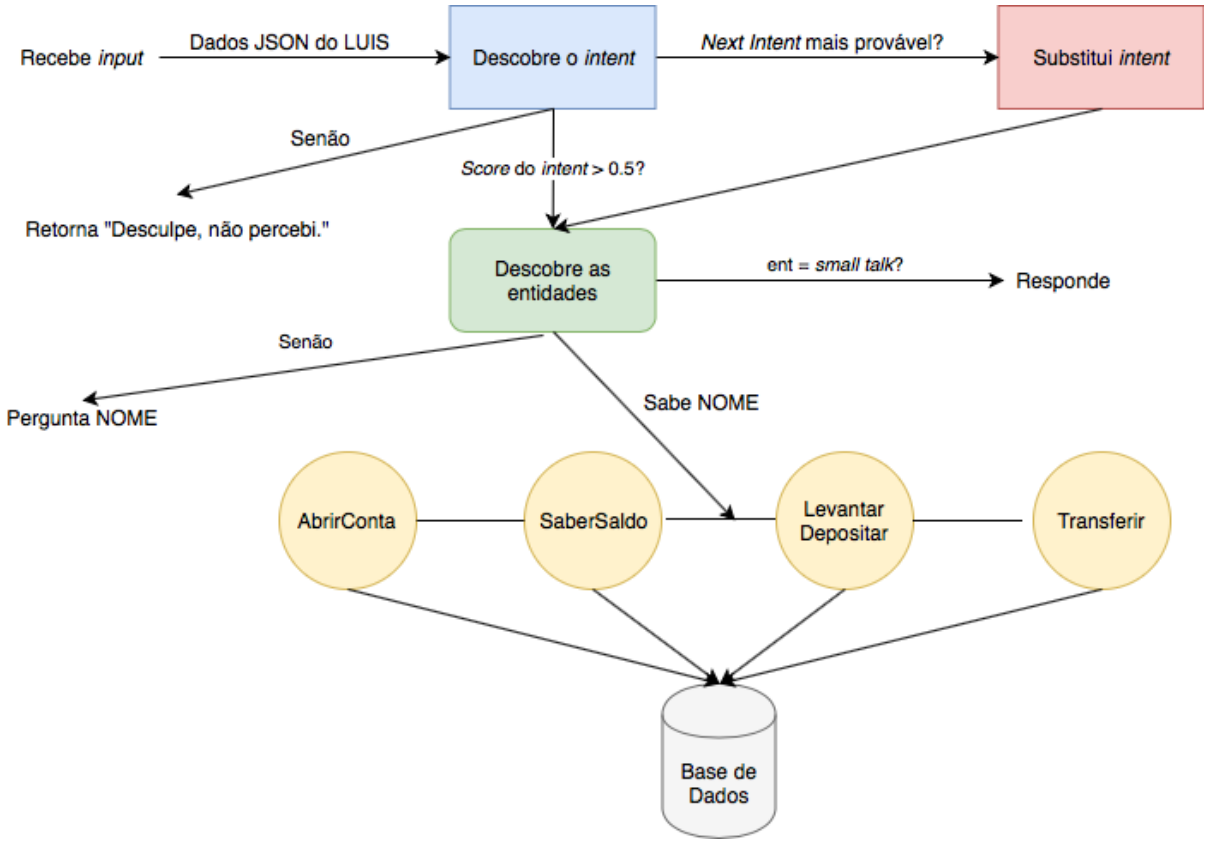


Figura 4.1: Arquitetura do *chatbot* proposto.

Foram criadas 12 intenções, 4 entidades e mais de 450 frases de diálogo para ajudar na classificação dos *intents*. De realçar que o LUIS, numa primeira fase, não era capaz de classificar automaticamente as *entities*, mas à medida que começou a receber informação também começou a saber reconhecer quais eram os valores das entidades.

Depois de treinada a aplicação, esta foi publicada, para que fizesse, de facto, algum sentido usar a plataforma. O LUIS fornece uma chave de subscrição juntamente com um *link* para utilização no *browser*, para que se adicione a *query* pretendida para a classificação.

Com isto foi possível ter, em formato JSON, informação vital para o funcionamento do *bot*, nomeadamente a intenção da *query* dada, as suas entidades e respetivos valores e até o *score* para o *intent* mais provável.

No entanto para receber estes dados de forma automática e sem que dependesse da utilização da API, foi necessário integrar o LUIS com o Azure, produto também da Microsoft.

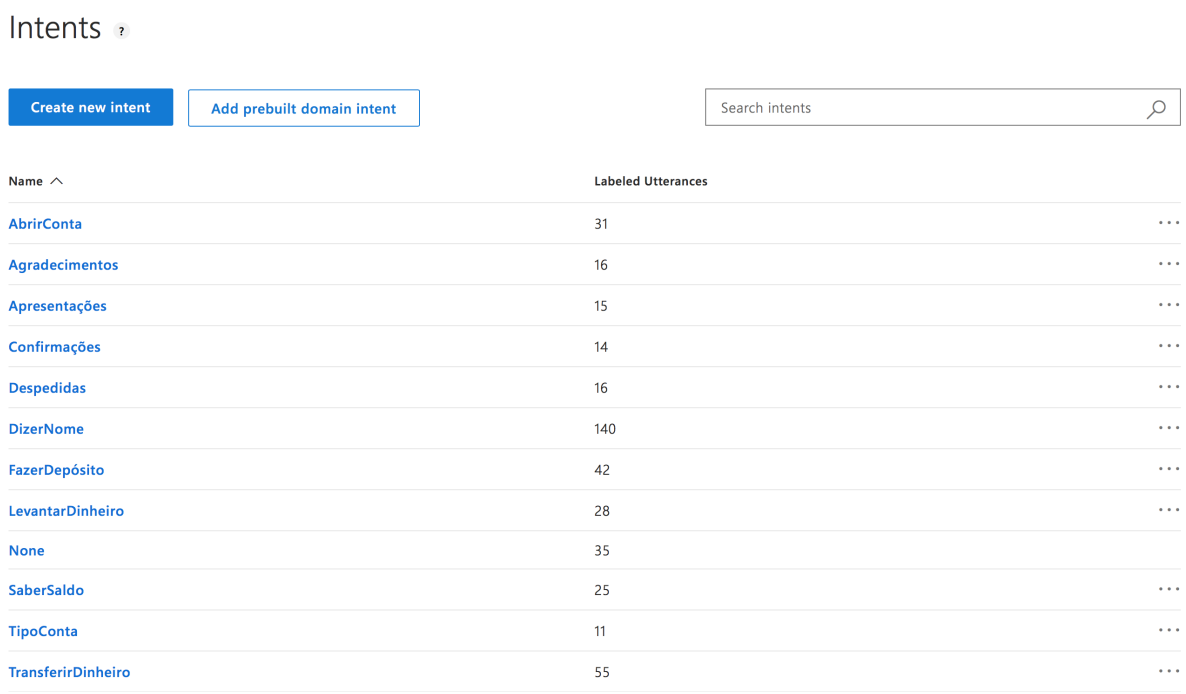


Figura 4.2: *Intents* criados no LUIS.

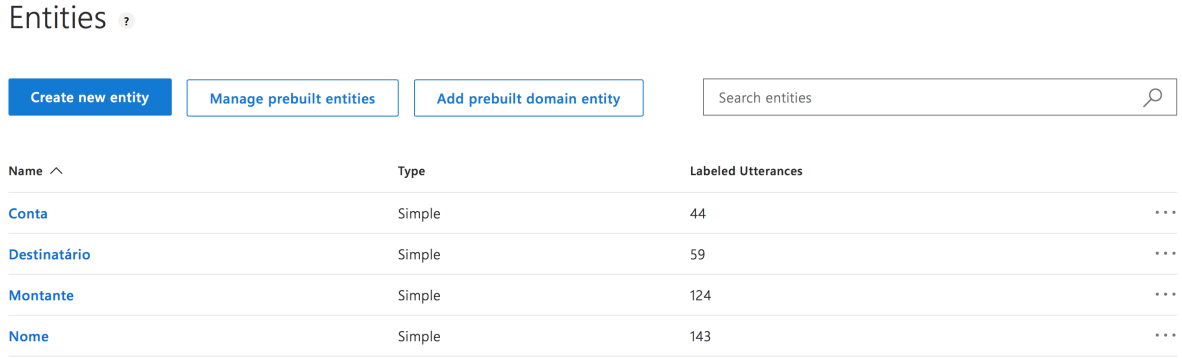


Figura 4.3: *Entities* criadas no LUIS.

4.2 Integração com o Azure

O Azure é uma plataforma de *cloud computing* utilizada para criar, testar, desenvolver e gerir aplicações e serviços através de uma rede global de *data centers* da Microsoft[28]. É também um grande sucesso junto das empresas, funcionando como SaaS (*Software as a Service*), com várias línguas, ferramentas e *software* disponíveis. Foi anunciado em outubro de 2008 na *Professional Developers Conference*, em Los Angeles, sob o nome de *Project Red Dog*, mas de lá para cá já mudou a sua designação para *Windows Azure* (em 2010) e mais recentemente *Microsoft Azure* (2014).

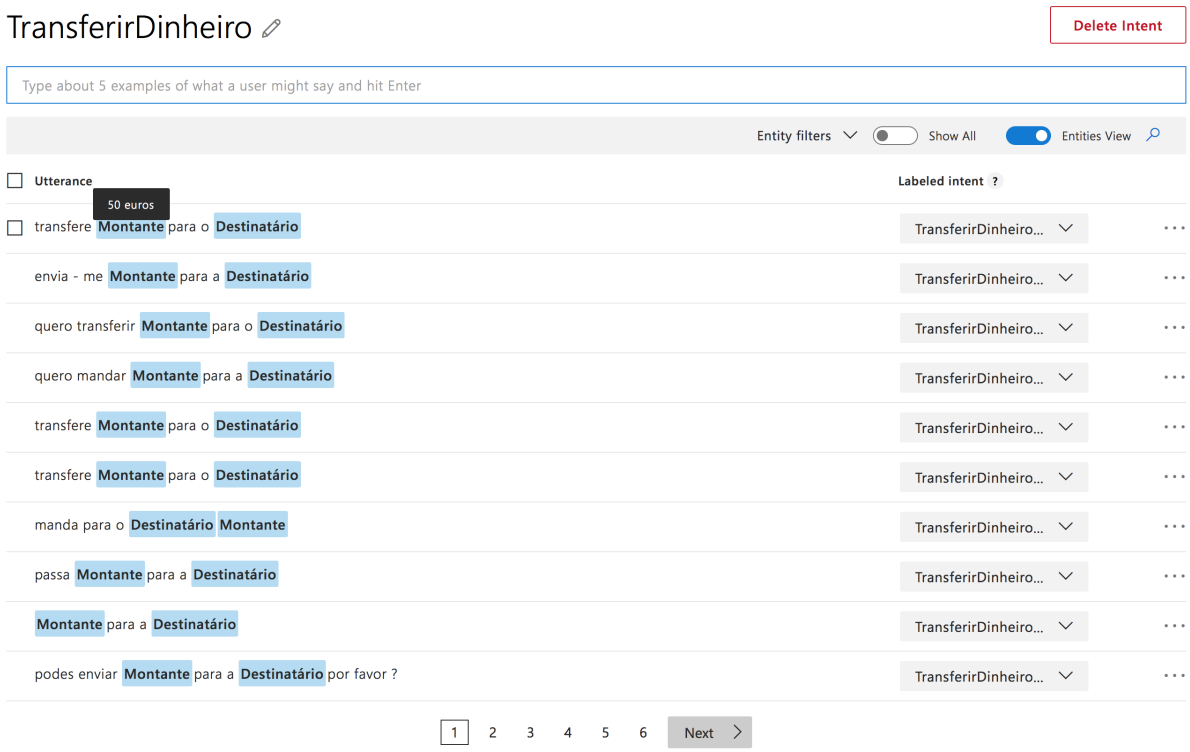


Figura 4.4: Exemplo de frases criadas no LUIS para o *intent* TransferirDinheiro e respetiva classificação de *entities*.

O portal do Azure permite a criação de vários recursos[29], entre eles o *Language Understanding*. Depois de configurar o nome, tipo de subscrição, localização, método de pagamento, etc., o portal fornece uma chave que deve ser adicionada na plataforma do LUIS. Assim, foi possível integrarar LUIS e Azure e conseguir chamar o *endpoint* através do código.

Para receber os dados em formato JSON usando o classificador do LUIS, foi necessário utilizar a chave de subscrição criada no Azure e adicionada no LUIS.

4.3 Código e Linguagem

A linguagem de programação utilizada foi o Python, até pela compatibilidade com a biblioteca `Chatterbot`, que foi utilizada numa primeira instância. No entanto, à medida que o trabalho foi mudando de rumo, nada impediu que se pudesse continuar a desenvolver o mesmo em Python, já que é uma linguagem instintiva e fácil de codificar.

A aplicação foi também desenvolvida em língua natural portuguesa, com os *intents*, *entities* e falas de diálogos a ser concebidos na língua paterna de Portugal, até por uma questão de interesse, já que há um maior valor acrescentado desta forma, ao invés do inglês.

```
{
  "query": "quero transferir 20 euros para a maria",
  "topScoringIntent": {
    "intent": "TransferirDinheiro",
    "score": 0.9964087
  },
  "intents": [
    {
      "intent": "TransferirDinheiro",
      "score": 0.9964087
    },
    {
      "intent": "FazerDeposito",
      "score": 0.0142978877
    },

    ...

    {
      "intent": "TipoConta",
      "score": 1.30989929E-05
    }
  ],
  "entities": [
    {
      "entity": "maria",
      "type": "Destinatario",
      "startIndex": 33,
      "endIndex": 37,
      "score": 0.9913221
    },
    {
      "entity": "20 euros",
      "type": "Montante",
      "startIndex": 17,
      "endIndex": 24,
      "score": 0.9991883
    }
  ]
}
```


Bloco de Código 4.1: Classificação para a *query* exemplo.

4.4 Diagrama de ficheiros

O programa conta com 11 ficheiros com funções distintas, mas que são dependentes uns dos outros. Para correr o *bot*, utiliza-se o ficheiro principal (`main.py`).

O **`main.py`** é o ficheiro que, com a chave de subscrição do Azure e os dados em formato

```
import requests

headers = {
    // Request headers
    'Ocp-Apim-Subscription-Key': 'YOUR-SUBSCRIPTION-KEY',
}

params ={
    // Query parameter
    'q': 'turn on the left light',
    // Optional request parameters, set to default values
    'timezoneOffset': '0',
    'verbose': 'false',
    'spellCheck': 'false',
    'staging': 'false',
}

try:
    r = requests.get('https://westus.api.cognitive.microsoft.com/
    luis/v2.0/apps/df67dcdb-c37d-46af-88e1-8b97951ca1c2',headers=headers,
        params=params)
    print(r.json())

except Exception as e:
    print("[Errno {0}] {1}".format(e.errno, e.strerror))
```

Bloco de Código 4.2: Receber dados JSON no código.

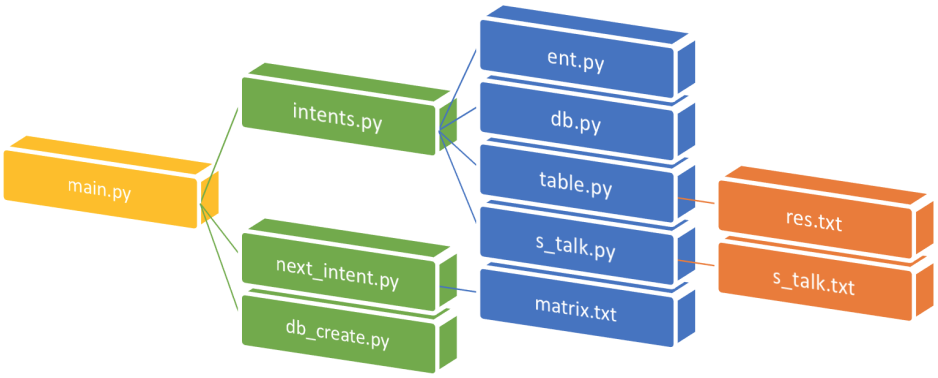


Figura 4.5: Ligações entre os ficheiros.

JSON do LUIS, faz o programa funcionar. Pede o *input* ao utilizador dentro de um ciclo `while` e depois usa esse *input* como *query* passada através do *link* que vai originar o JSON. Desse código em formato JSON retira o `topScoringIntent` e o seu *score*. O `main` chama também o ficheiro **`db_create.py`** que cria as tabelas *Utilizador* e *Conta* em *pythonsqlite.db*, se elas não existirem. A tabela *Utilizador* conta como entradas o *id* e o *nome* da pessoa que tem a conta bancária e a tabela *Conta* lista o tipo de conta, o montante e o *id* do utilizador.

```
$ python3 main.py
```

Bloco de Código 4.3: Compilar *bot*.

```
// Intent da query
intent = r.json()['topScoringIntent']['intent']
// Score do intent
score = r.json()['topScoringIntent']['score']
```

Bloco de Código 4.4: Excerto do `main`.

O ficheiro **`intents.py`** trata das intenções. Faz a sua operação ou determina o diálogo consoante o *intent* que lhe é passado.

O **`next_intent.py`** utiliza uma série de ligações entre os *intents* e os *intents* seguintes mais prováveis (contida no ficheiro **`matrix.txt`**), junto com a sua probabilidade (de notar que a soma tem de dar sempre 1), para determinar se a intenção seguinte está mais próxima de 1 do que o *score* da classificação do *input* que o utilizador vai dar. Se for, substitui o *intent* pelo *next_intent*.

O ficheiro `intents.py` tem uma série de ligações a outros ficheiros, que estão importados no seu código. O ficheiro **`ent.py`** reconhece e extrai as entidades através do JSON que lhe é passado; o ficheiro **`db.py`** adiciona os valores às tabelas sempre que é feita uma operação (por exemplo abrir uma conta em nome de alguém); o ficheiro **`s_talk.py`** utiliza os *intents* de *small talk* contidos no ficheiro de texto **`s_talk.txt`** para passá-los para uma lista; e o ficheiro **`table.py`** faz o mesmo com outro ficheiro de texto, mas desta feita o ficheiro **`res.txt`**, que engloba os diálogos que não vão precisar de tratamento e são de aplicação direta para esses mesmos *intens* de *small talk* e seleciona de forma aleatória numa das falas para responder.

4.5 *Flow* do Programa

O *flow* do programa leva o utilizador a seguir um certo tipo de caminhos que são mais vantajosos ao *developer*[30]. Algo que foi tido em conta neste trabalho.

Quando se corre o ficheiro `main.py` o *bot* começa por dizer ***Sou um bot construído para realizar serviços bancários (abrir uma conta, consultar o saldo, realizar transferências). Diga-me o seu nome, por favor.***, o que o instiga o utilizador a duas coisas. A primeira é que proporciona a sensação que faz um conjunto limitado de tarefas e ao listá-las, de certa forma previne o utilizador de perguntar coisas muito distantes do que ele faz. Ou seja, o utilizador não vai pensar que o *bot* tem funções infinitas e que serve, propositadamente, para aquilo (também o nome do *bot* - ChatBanco - não leva a que se confunda com uma pessoa). A segunda é que lhe pede logo o nome, o que será importante para as tarefas seguintes, como a

Tabela 4.1: *Intents* seguintes.

Intent	*Next Intent*	Probabilidade
AbrirConta	TipoConta	0.7
	DizerNome	0.3
Agradecimentos	SaberSaldo	0.45
	TransferirDinheiro	0.55
Apresentações	TransferirDinheiro	0.2
	AbrirConta	0.3
	SaberSaldo	0.5
Confirmações	SaberSaldo	0.4
	Agradecimentos	0.6
Despedidas	AbrirConta	0.3
	SaberSaldo	0.4
	TransferirDinheiro	0.3
DizerNome	Agradecimetnos	0.6
	FazerDepósito	0.4
FazerDepósito	SaberSaldo	0.3
	Confirmações	0.7
LevantarDinheiro	SaberSaldo	0.3
	Confirmações	0.7
SaberSaldo	TransferirDinheiro	0.2
	Agradecimentos	0.1
	DizerNome	0.7
TipoConta	FazerDepósito	0.6
	Agradecimentos	0.4
TransferirDinheiro	SaberSaldo	0.3
	Confirmações	0.7
None	AbrirConta	0.3
	SaberSaldo	0.4
	TransferirDinheiro	0.3

abertura da conta ou até a procura na base de dados pelos seus dados da conta.

Mas mesmo que o utilizador não dê logo o seu nome, quando quiser fazer alguma operação em que seja necessário fornecer o nome, o *bot* é capaz de perceber ou não se o utilizador já lhe disse o seu nome e, em caso negativo, perguntar-lhe. Assim, quando o utilizador quiser abrir uma conta, mas ainda não partilhou o seu nome, será retornada a resposta ***Diga-me o seu nome, por favor.***. O mesmo acontecerá quando quiser saber o seu saldo ou depositar, levantar e transferir dinheiro.

O *flow* do *bot* também está presente quando o `next_intent` suplanta a classificação do próprio *intent*, uma vez que leva o utilizador para a intenção que o programa está à espera.

Tabela 4.2: Exemplo de respostas para os *intents* de *small talk*.

Intent	*Falas*
Agradecimentos	De nada! É a minha função. O prazer é meu.
Apresentações	Olá, seja bem-vindo! Olá, como posso ajudá-lo? Olá, é bom vê-lo de novo.
Confirmações	Vou fazer isso por si.
Despedidas	Estarei aqui quando voltar. Eu fico por aqui! Voltaremos a ver-nos.
None	Em que posso ajudar?

Por fim, quando o *input* não está próximo de nenhum *intent* e em que o *score* não passa dos 50%, é retornada a resposta ***Desculpe, não percebi.***.

4.6 Base de dados

Para tornar o *bot* mais sofisticado utilizou-se uma base de dados com duas tabelas, *Conta* e *Utilizador*, criada como um serviço adicional. Quando o utilizador pede ao programa para abrir uma conta, e lhe passa o seu nome e tipo de conta, é criada uma instância na tabela *Conta* com o tipo da conta, 0 euros e o utilizador_id que está referenciado na tabela *Utilizador* com o seu nome.

```
sqlite> select * from conta;
id          tipo        montante    utilizador_id
----------  ----------  ----------  -------------
1           corrente    0           1

sqlite> select * from utilizador;
id          nome
----------  ----------
1           tiago
```

Figura 4.6: Exemplo de entrada na base de dados.

4.6.1 Criação

O *bot* conta com dois ficheiros que realizam operações em SQLite, o `db_create.py` e o `db.py`. O primeiro, como já vimos, é chamado no `main.py` para criar o ficheiro *pythonsqlite.db* com as tabelas *Conta* e *Utilizador* se ainda não existirem.

```
def main():
    database = "pythonsqlite.db"
    sql_create_user_table = """CREATE TABLE IF NOT EXISTS utilizador (
                                    id integer PRIMARY KEY,
                                      nome text NOT NULL); """
    sql_create_conta_table = """CREATE TABLE IF NOT EXISTS conta (
                                    id integer PRIMARY KEY,
                                    tipo text NOT NULL,
                                    montante integer NOT NULL,
                                    utilizador_id integer NOT NULL,
                                    FOREIGN KEY (utilizador_id) REFERENCES
                                        utilizador (id));"""
```

Bloco de Código 4.5: Criar base de dados e tabelas.

4.6.2 *Update*

Já o segundo contém as funções de inserção de dados nas tabelas e outras de acesso à informação do saldo e de *update* de valores, quando for pretendido movimentar dinheiro. É chamado quando o *intent* for algum que tenha associado a si uma operação, como AbrirConta, SaberSaldo, TransferirDinheiro, Fazer Depósito ou LevantarDinheiro.

```
database = "pythonsqlite.db"
conn = create_connection(database)
    with conn:
       sql = ''' INSERT OR IGNORE INTO utilizador(nome) VALUES(?) '''
       cur = conn.cursor()
       cur.execute(sql, (name,))
       return cur.lastrowid
```

Bloco de Código 4.6: Criar utilizador.

```
database = "pythonsqlite.db"

conn = create_connection(database)
    with conn:
       sql = ''' INSERT INTO conta(tipo,montante,utilizador_id) VALUES(?,?,?)
           '''
       cur = conn.cursor()
       cur.execute(sql, (tipo,montante,create_user(name),))
       return cur.lastrowid
```

Bloco de Código 4.7: Criar conta.

```
database = "pythonsqlite.db"

conn = create_connection(database)
   with conn:
      cur = conn.cursor()
      cur.execute("""
         SELECT DISTINCT montante FROM utilizador, conta WHERE utilizador_id =
             (SELECT utilizador.id FROM utilizador, conta WHERE nome = ?); """,
             (nome,))
      sal = cur.fetchone()
      if sal is None:
         return 0
      else:
         return str(sal).strip("()").strip(",")
```

Bloco de Código 4.8: Saber saldo.

```
database = "pythonsqlite.db"

conn = create_connection(database)
   with conn:
      cur = conn.cursor()
      cur.execute("""UPDATE conta SET montante = ? WHERE utilizador_id =
          (SELECT utilizador.id FROM utilizador, conta WHERE nome = ?);""",
          (montante, nome,))
```

Bloco de Código 4.9: Atualizar valores.

4.6.3 Operações

Como vimos em cima, existem 5 *intents* que fazem operações relacionadas com a base de dados. Vejamos o funcionamento de cada uma destas intenções e a sua ligação com as tabelas da BD:

AbrirConta: Se o utilizador disser que pretende abrir uma conta e referenciar o seu nome e o tipo da conta, será chamada a função `create_account` do ficheiro `db.py`, passando o tipo de conta, o montante como 0 e o nome.

SaberSaldo: Se o utilizador quiser saber o saldo e tiver partilhado o seu nome, é chamada a função `saber_saldo` do ficheiro `db.py`, passando o nome como argumento.

FazerDepósito: Se o utilizador quiser fazer um depósito e tiver partilhado o seu nome, primeiro vai-se buscar o seu saldo com o auxílio da função `saber_saldo`, adiciona-se a esse saldo o montante do depósito e depois substitui-se na tabela através da função *atualizar_valores*, com o seu nome e novo montante.

LevantarDinheiro: Se o utilizador quiser levantar dinheiro e tiver partilhado o seu nome, primeiro vai-se buscar o seu saldo com o auxílio da função `saber_saldo`, subtrai-se a esse saldo

o montante do depósito (se o montante que quiser levantar for menor), e depois substitui-se na tabela através da função *atualizar_valores*, com o seu nome e novo montante. Se o saldo que tiver não for suficiente para o valor que o utilizador quer levantar, é retornada a resposta ***Não tem dinheiro suficiente na Conta.***

TransferirDinheiro: Se o utilizador quiser transferir dinheiro para outro utilizador e tiver partilhado o seu nome, primeiro vai-se buscar o seu saldo e o da pessoa a quem quer enviar dinheiro (utiliza-se como nome a entidade *destinatário*) com o auxílio da função `saber_saldo`, subtrai-se a esse saldo o montante que deseja transferir (se o montante que quiser levantar for menor) e ao do destinatário faz-se o mesmo, mas somando, e depois substitui-se nas respetivas entradas na tabela através da função *atualizar_valores*, com o nome e destinatário e os novos montantes. Se o saldo que tiver não for suficiente para o valor que o utilizador quer levantar, é retornada a resposta ***Não tem dinheiro suficiente na Conta.***

4.7 Opções de configuração

Os ficheiros de texto com frases são umas das opções de personalização acessíveis ao *bot*. A configuração destes elementos é de fácil acesso e poderá ser incluída em trabalho futuro.

- `matrix.txt` - *Intents* seguintes e probabilidade para cada *intent*
- `s_talk.txt` - *Small talk*
- `res.txt` - Diálogos para cada *intent* de *small talk*

4.8 Descrição da solução

Para além do desenvolvimento do protótipo, são ainda apresentadas as funções utilizadas em pseusdo-código, para que o utilizador perceba em alto-nível não só como foram feitas, mas também como podem ser adaptadas para outros projetos envolvendo *chatbots*.

4.9 Repositório no GitHub

Todos os ficheiros do trabalho, juntamente com instalação e funcionamento do *bot* podem ser consultados em: https://github.com/fabioteixeira2up/chatbot.git.

```
IMPORTS
USAR CHAVE DE SUBSCRICAO
ENQUANTO FOR VERDADE
   LER INPUT
   ADICIONAR INPUT AOS PARAMETROS
   TENTAR
      RECEBER REQUEST DA QUERY
      DESCOBRIR O INTENT DA QUERY
      DESCOBRIR O SCORE DO INTENT DA QUERY
      VER SE O PROXIMO INTENT GERADO TEM MAIS PRECISAO QUE O ATUAL
      CHAMAR FUNCAO COM OS DIALOGOS DOS INTENTS
      VER O INTENT SEGUINTE
   SENAO ERRO
```

Bloco de Código 4.10: Pseudo-código (`main.py`).

```
IMPORTS
FUNCAO
   SE INTENT EM S_TALK
      TABLE(INTENT)
   SENAO SE INTENT EM DIZER_NOME
      IMPRIMIR
   SENAO SE INTENT EM ABRIR_CONTA
      SE NOME == ""
         IMPRIMIR
      SENAO
         CREATE_ACCOUNT(CONTA, 0, NOME)
         IMPRIMIR
   SENAO SE INTENT EM SABER_SALDO
      SE NOME == ""
         IMPRIMIR
      SENAO
         SABER_SALDO(NOME)
         IMPRIMIR
   SENAO SE INTENT EM FAZER_DEPOSITO
      SE NOME == ""
         IMPRIMIR
      SENAO
         ATUALIZAR_VALORES(NOME, SABER_SALDO(NOME) + MONTANTE)
         IMPRIMIR
   SENAO SE INTENT EM LEVANTAR_DINHEIRO
      SE NOME == ""
         IMPRIMIR
      SENAO
         SALDO = SABER_SALDO(NOME)
         SE SALDO >= MONTANTE
            ATUALIZAR_VALORES(NOME, SABER_SALDO(NOME) - MONTANTE)
            IMPRIMIR
         SENAO
            IMPRIMIR
   SENAO SE INTENT EM TRANSFERIR_DINHEIRO
      SE NOME = ""
         IMPRIMIR
      SENAO
         SALDO1 = SABER_SALDO(NOME)
         SALDO2 = SABER_SALDO(DESTINATARIO)
         SE SALDO1 >= MONTANTE
            ATUALIZAR_VALORES(NOME, SALDO1 - MONTANTE)
            IMPRIMIR
            ATUALIZAR_VALORES(DESTINATARIO, SALDO2 + MONTANTE)
         SENAO
            IMPRIMIR
   SENAO
      IMPRIMIR
```

Bloco de Código 4.11: Pseudo-código (`intents.py`).

```
IMPORTS
FUNCAO
   ABRIR FICHEIRO MATRIX.TXT COM NEXT_INTENT E PROBABILIDADES
   CRIAR LISTAS COM DADOS DO FICHEIRO
   FILTRAR POSICOES VAZIAS
   ESCOLHER ALEATORIAMENTE COM BASE NAS PROBABILIDADES
   RETORNAR NEXT_INTENT E A SUA PROBABILIDADE
```

Bloco de Código 4.12: Pseudo-código (`next_intent.py`).

```
IMPORTS
ABRIR FICHEIRO RES.TXT COM DIALOGOS POR INTENT
FUNCAO
   CRIAR LISTAS COM DADOS DO FICHEIRO
   ESCOLHER ALEATORIAMENTE FALA PARA O INTENT
   IMPRIMIR
```

Bloco de Código 4.13: Pseudo-código (`table.py`).

```
FUNCAO
   SE ENTIDADES != ""
      PARA CADA ENTIDADE
         ASSOCIAR ENTIDADES AS VARIAVEIS MONTANTE, CONTA, DESTINATARIO, NOME
   RETORNAR ENTIDADES
```

Bloco de Código 4.14: Pseudo-código (`ent.py`).

```
FUNCAO
   ABRIR FICHEIRO S_TALK.TXT COM OS INTENTS DE SMALL TALK
   ENVIAR OS INTENTS PARA UMA LISTA
   RETORNAR A LISTA
```

Bloco de Código 4.15: Pseudo-código (`s_talk.py`).

Capítulo 5

Avaliação e Demonstração

Este capítulo aborda a demonstração da utilização do *chatbot* proposto (o ChatBanco), bem como a avaliação quantitativa da solução. Adicionalmente é exibido o valor acrescentado, juntamente com uma colaboração, e são revistos alguns testes e experiências abordadas.

5.1 Demonstração de utilização

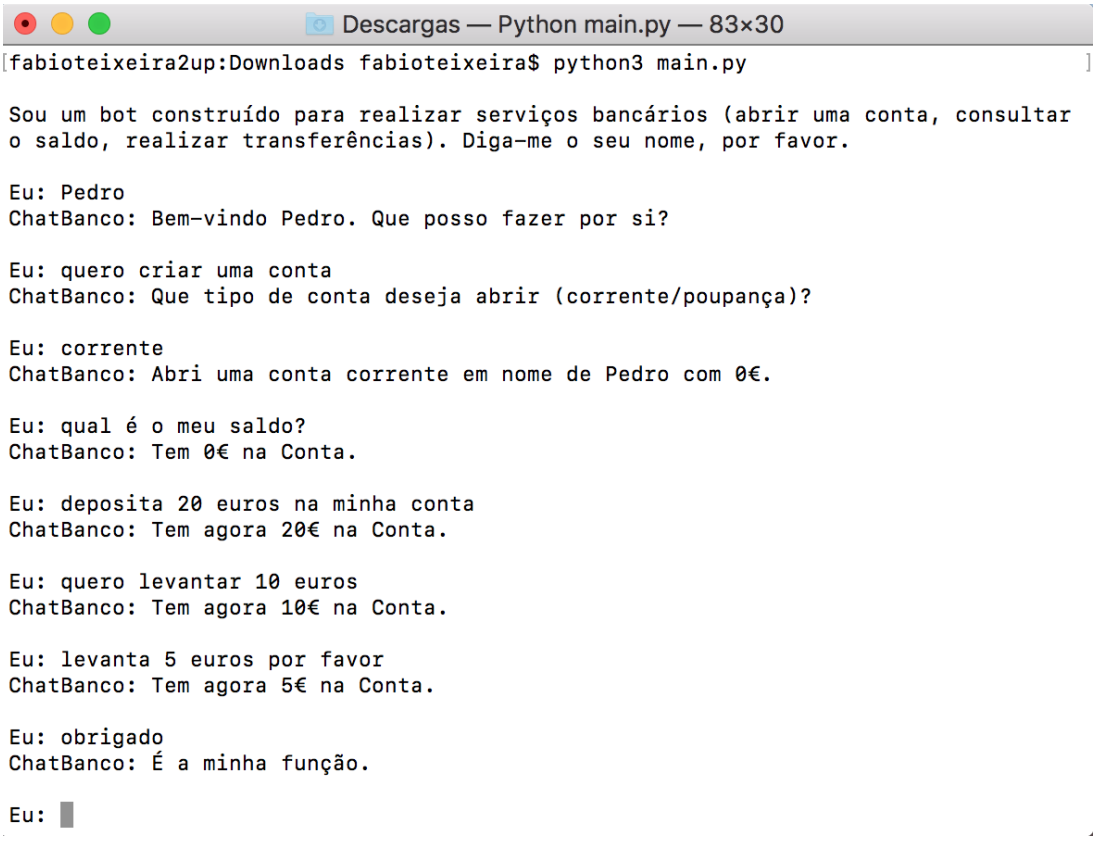


Figura 5.1: Exemplo de um diálogo bancário com o *chatbot* proposto.

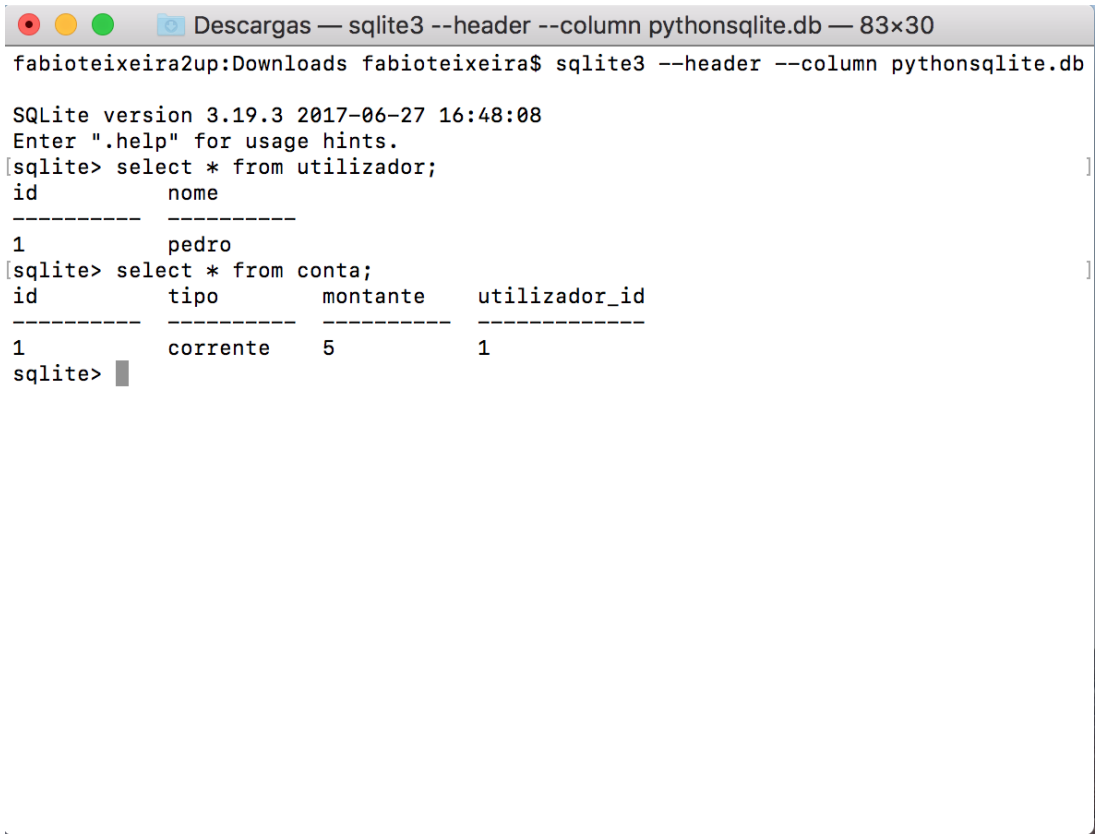


Figura 5.2: Entradas criadas na base de dados depois da conversa.

A solução proposta atingiu os objetivos, na medida que em possibilita ao utilizador criar uma conta, saber o saldo, depositar, levantar ou transferir dinheiro, ao mesmo tempo que mantém uma interface limpa, simplista e de fácil interação. Como podemos ver no exemplo demonstrado, o utilizador Pedro decide abrir uma conta do tipo corrente, que naturalmente começa com 0 euros, e depois efetua uma série de operações sobre o seu saldo, que vão atualizando os seus dados na tabela de entradas. No final da conversa, o Pedro ficou com 5 euros na sua conta corrente, como pode ser observado na base de dados.

5.2 Valor acrescentado

O valor acrescentado do projeto está, claro, na criação de um *bot* bancário em língua portuguesa, nas intenções e entidades definidas no LUIS, nas mais de 450 falas de diálogo, nos ficheiros de texto com *intents*, no *flow* do programa e ligação entre os ficheiros que obriga o utilizador a ir por um certo caminho, na definição dos *intents* seguintes, nas respostas para cada *intent*, nas funções definidas e na base de dados que simula a relação entre consumidor e banco.

5.3 Colaboração com a documentação do **Chatterbot**

Apesar dos vários ficheiros de texto criados para o *dataset* da biblioteca `Chatterbot` não terem tido propriamente implicações no *bot* final, estes serão utilizados como colaboração para a documentação do *corpus* do `Chatterbot` no GitHub (https://github.com/gunthercox/chatterbot-corpus/tree/master/chatterbot_corpus/data).

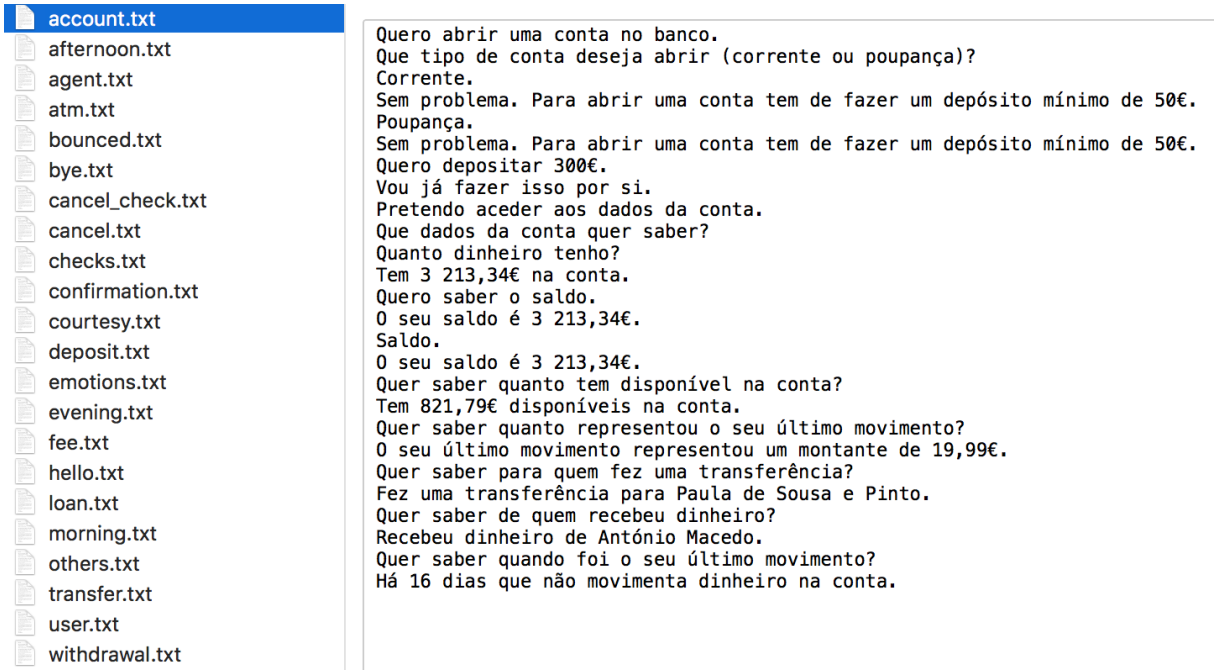


Figura 5.3: Ficheiros do *dataset* de diálogos.

5.4 Experiências e Testes

5.4.1 LUIS vs. Watson

Ao longo da realização do trabalho foram abordadas e experimentas três formas para concluir o objetivo de construir um *chatbot*. A utilização da biblioteca `Chatterbot` acabou por revelar-se insuficiente para o nível de sofisticação pretendido, uma vez que não se comportava como NLP, mas antes NLU, e o *bot* não ganhava a condição de conseguir decidir por ele próprio, tendo, antes, de ser abordados os casos de forma unitária.

Também a criação, manualmente, de *intents* não foi a estratégia escolhida, apesar de ter sido trabalhada durante algum tempo, porque seria difícil construir um classificador de *intents* que chegasse perto do nível das plataformas já existentes. E também porque, na extração de *entities* seria complicado definir os valores, uma vez que estes eram imprevisíveis, ao invés de serem apenas um *match* de valores previamente definidos.

A utilização de uma API como o LUIS e a sua integração com o Azure foi então a opção encontrada que melhor servia os requisitos. No entanto, aqui surgiu uma espécie de problema que foi o período de subscrição gratuito. Uma vez finalizado esse tempo foi necessário tentar encontrar outra plataforma para desempenhar esse papel ou ativar um plano de pagamento.

Para isso testou-se o Watson, da IBM. Esta API também permite a classificação de *intents* e *entities*, juntamente com a criação de diálogos para cada intenção[31]. No entanto, contrariamente ao LUIS, tem uma grande desvantagem que se prende com a obrigatoriedade de ter de definir os valores para as entidades antes do reconhecimento dos mesmos. Algo que não era de todo viável para o trabalho. Vejamos o seguinte exemplo:

Transfere 50 euros para o João

Na frase acima exposta as entidades e os seus valores são:

Montante: 50 euros; **Destinatário**: João.

No entanto para que o Watson reconheça que 50 euros é o valor da entidade *Montante* tem de se adicionar o valor 50 euros aquando da criação das entidades. Agora imaginemos que o valor é 120 euros, 200 euros, 40 euros, etc. Imaginemos também que o valor do nome, ao invés de João, é Margarida, ou Paulo, ou Ana... Ou seja, o âmbito de um *bot* que utiliza o classificador Watson, da IBM, não consegue suportar a lista de todos os valores possíveis que o utilizador pode referenciar.

O LUIS é capaz de extrair entidades dinamicamente, enquanto que o Watson executa a extração através de uma lista predefinida de valores, já que é necessário fornecer o conjunto completo de valores possíveis para cada entidade. Por estas razões, o Watson deixou de ser uma hipótese.

A solução encontrada foi manter a subscrição no Azure e definir um método de pagamento *pay-as-you-go* para utilizar os recursos pretendidos, nomeadamente o *bot framework* LUIS.

5.4.2 Testes quantitativos

Para testar a fiabilidade do reconhecimento de intenções e entidades, listaram-se 10 frases de diálogo centradas em serviços bancários, apresentadas de seguida. Podemos observar que o *score* mínimo na classificação é 0.7767221, o que prova que as 10 frases obtiveram, de largo modo, sucesso e que o *bot* recebe os dados certos para fornecer ao utilizador em modo de diálogo.

Em baixo podemos ver os resultados dos testes de duas formas, a primeira exibida sob a forma de uma tabela, com as frases e *intent* e *score* respetivos; já a segunda lista os casos de forma unitária, com maior ênfase no detalhe. No último caso é possível constatar também as entidades e valores de precisão para os seus valores.

5.4.2.1 Vista geral dos casos de uso

Tabela 5.1: Testes quantitativos.

Frase	*Intent*	*Score*
Quero abrir uma conta	AbrirConta	0.998173
Abre uma conta corrente	AbrirConta	0.8736825
Quero abrir uma conta poupança	AbrirConta	0.987042964
Diz-me o meu saldo	SaberSaldo	0.976276934
Quanto dinheiro tenho na conta	SaberSaldo	0.9569219
Faz um depósito de 220 euros	FazerDepósito	0.957191348
Quero depositar 50 euros	FazerDepósito	0.9710937
Levanta 30 euros	LevantarDinheiro	0.7767221
Quero tirar 55 euros da conta	LevantarDinheiro	0.867065847
Transfere 130 euros para a Maria	TransferirDinheiro	0.995374441

5.4.2.2 Vista detalhada dos casos de uso

```
{
  "query": "Quero abrir uma conta",
  "topScoringIntent": {
    "intent": "AbrirConta",
    "score": 0.998173
  },
  "entities": []
}
```

Bloco de Código 5.1: *Quero abrir uma conta.*

```
{
  "query": "Abre uma conta corrente",
  "topScoringIntent": {
    "intent": "AbrirConta",
    "score": 0.8736825
  },
  "entities": [
    {
      "entity": "corrente",
      "type": "Conta",
      "startIndex": 15,
      "endIndex": 22,
      "score": 0.996424
    }
  ]
}
```


Bloco de Código 5.2: *Abre uma conta corrente.*


```
{
  "query": "Quero abrir uma conta poupan a",
  "topScoringIntent": {
    "intent": "AbrirConta",
    "score": 0.987042964
  },
  "entities": [
    {
      "entity": "poupan a",
      "type": "Conta",
      "startIndex": 22,
      "endIndex": 29,
      "score": 0.994411767
    }
  ]
}
```


Bloco de Código 5.3: *Quero abrir uma conta poupança.*


```
{
  "query": "Diz-me o meu saldo",
  "topScoringIntent": {
    "intent": "SaberSaldo",
    "score": 0.976276934
  },
  "entities": []
}
```


Bloco de Código 5.4: *Diz-me o meu saldo.*

```
{
  "query": "Quanto dinheiro tenho na conta?",
  "topScoringIntent": {
    "intent": "SaberSaldo",
    "score": 0.9569219
  },
  "entities": []
}
```


Bloco de Código 5.5: *Quanto dinheiro tenho na conta?.*


```
{
  "query": "Faz um dep sito de 220 euros",
  "topScoringIntent": {
    "intent": "FazerDep sito",
    "score": 0.957191348
  },
  "entities": [
    {
      "entity": "220 euros",
      "type": "Montante",
      "startIndex": 19,
      "endIndex": 27,
      "score": 0.9922509
    }
  ]
}
```


Bloco de Código 5.6: *Faz um depósito de 220 euros.*


```
{
  "query": "Quero depositar 50 euros",
  "topScoringIntent": {
    "intent": "FazerDep sito",
    "score": 0.9710937
  },
  "entities": [
    {
      "entity": "50 euros",
      "type": "Montante",
      "startIndex": 16,
      "endIndex": 23,
      "score": 0.9957119
    }
  ]
}
```


Bloco de Código 5.7: *Quero depositar 50 euros.*

```
{
  "query": "Levanta 30 euros",
  "topScoringIntent": {
    "intent": "LevantarDinheiro",
    "score": 0.7767221
  },
  "entities": [
    {
      "entity": "30 euros",
      "type": "Montante",
      "startIndex": 8,
      "endIndex": 15,
      "score": 0.9933681
    }
  ]
}
```


Bloco de Código 5.8: *Levanta 30 euros.*


```
{
  "query": "Quero tirar 55 euros da conta",
  "topScoringIntent": {
    "intent": "LevantarDinheiro",
    "score": 0.867065847
  },
  "entities": [
    {
      "entity": "55 euros",
      "type": "Montante",
      "startIndex": 12,
      "endIndex": 19,
      "score": 0.997447431
    }
  ]
}
```


Bloco de Código 5.9: *Quero tirar 55 euros da conta.*

```
{
  "query": "Transfere 130 euros para a Maria",
  "topScoringIntent": {
    "intent": "TransferirDinheiro",
    "score": 0.995374441
  },
  "entities": [
    {
      "entity": "maria",
      "type": "Destinat rio",
      "startIndex": 27,
      "endIndex": 31,
      "score": 0.991374969
    },
    {
      "entity": "130 euros",
      "type": "Montante",
      "startIndex": 10,
      "endIndex": 18,
      "score": 0.997410357
    }
  ]
}
```


Bloco de Código 5.10: *Transfere 130 euros para a Maria.*

Capítulo 6

Conclusões

O mundo dos agentes automáticos de conversação pode trazer significantes melhorias às empresas, permitindo um atendimento instantâneo e rápido e uma interface similar à das mensagens de texto que toda a gente sabe utilizar. E isto tudo numa tecnologia que evolui de dia para dia e que é aprimorada de forma contínua.

Os objetivos principais consistiam em estudar a forma como os *chatbots* têm sido apresentados ao longo dos tempos, analisar o estado corrente da tecnologia e o que tem sido feito na área, procurar ferramentas existentes para possível adaptação futura e, por fim, produzir um protótipo demonstrável tão sofisticado quando possível, o que foi atingido.

O *chatbot* foi contruído, sobretudo, sob uma interface em língua natural (português) para acesso à informação da conta. Numa segunda instância começou a efetuar outras operações como transferências, levantamentos ou depósitos de dinheiro. E posteriormente adotou a possibilidade de se ligar a uma base de dados SQL, que integrava os dados de que a aplicação se iria servir para responder ao utilizador.

O resultado final foi um protótipo de um *chatbot* bancário em língua portuguesa, de nome ChatBanco, programado em Python, com utilização de bases de dados em SQLite e ainda de um *bot framework* auxiliar. O *bot* é configurável para outros temas que não a área bancária, é de fácil utilização e manutenção e garante rapidez (praticamente instantâneo) na geração de respostas. O trabalho conta ainda com:

- Mais de 450 falas de diálogo treinadas no *bot framework*;
- Cerca de 15 intenções e entidades definidas no LUIS;
- Ficheiros de texto programáveis com respostas geradas para cada intenção;
- Funções para criar uma conta, saber o saldo e depositar, levantar ou transferir dinheiro;
- Base de dados que simula os dados bancários extraídos da conversa após a criação de uma conta.

6.1 Trabalho Futuro

Futuramente, o plano é continuar a desenvolver as conversações e serviços, especialmente a parte focada da área bancária, aumentar o conjunto de frases para os *intents* de *small_talk* e, se possível, lançar o *bot* numa plataforma de *messaging*, como o Messenger do Facebook.

Bibliografia

[1] Karl Branting, James Lester, and Bradford Mott. Dialogue management for conversational case-based reasoning. In *European Conference on Case-Based Reasoning*, pages 77–90. Springer, 2004.

[2] Stuart Russell, Peter Norvig, and Artificial Intelligence. A modern approach. *Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs*, 25:27, 1995.

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.

[4] Crm strategies and technologies to understand, grow and manage customer experiences. https://www.gartner.com/imagesrv/summits/docs/na/customer-360/C360_2011_brochure_FINAL.pdf. Accessed: 2018-01-13.

[5] Donald J Stoner, Louis Ford, and Mark Ricci. Simulating military radio communications using speech recognition and chat-bot technology. *The Titan Corporation, Orlando*, 2004.

[6] Terry Winograd. Understanding natural language. *Cognitive psychology*, 3(1):1–191, 1972.

[7] Maria Vargas-Vera and Miltiadis D Lytras. Aqua: A closed-domain question answering system. *Information Systems Management*, 27(3):217–225, 2010.

[8] Silvia Quarteroni and Suresh Manandhar. Designing an interactive open-domain question answering system. *Natural Language Engineering*, 15(1):73–95, 2009.

[9] Elena Cabrio, Julien Cojan, Alessio Palmero Aprosio, Bernardo Magnini, Alberto Lavelli, and Fabien Gandon. Qakis: an open domain qa system based on relational patterns. In *International Semantic Web Conference, ISWC 2012*, 2012.

[10] Ming-Hsiang Su, Chung-Hsien Wu, Kun-Yi Huang, Qian-Bei Hong, and Hsin-Min Wang. A chatbot using lstm-based multi-layer embedding for elderly care. In *Orange Technologies (ICOT), 2017 International Conference on*, pages 70–74. IEEE, 2017.

[11] Lorenz Cuno Klopfenstein, Saverio Delpriori, Silvia Malatini, and Alessandro Bogliolo. The rise of bots: a survey of conversational interfaces, patterns, and paradigms. In *Proceedings of the 2017 Conference on Designing Interactive Systems*, pages 555–565. ACM, 2017.

[12] Margaret-Anne Storey and Alexey Zagalsky. Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 928–931. ACM, 2016.

[13] Asbjørn Følstad and Petter Bae Brandtzæg. Chatbots and the new world of hci. *interactions*, 24(4):38–42, 2017.

[14] Victor W Zue and James R Glass. Conversational interfaces: Advances and challenges. *Proceedings of the IEEE*, 88(8):1166–1180, 2000.

[15] Stefan Bieliauskas and Andreas Schreiber. A conversational user interface for software visualization. In *Software Visualization (VISSOFT), 2017 IEEE Working Conference on*, pages 139–143. IEEE, 2017.

[16] Meet 11 of the most interesting chatbots in banking. https://thefinancialbrand.com/71251/chatbots-banking-trends-ai-cx/. Accessed: 2018-07-07.

[17] Edward Loper and Steven Bird. NLTK: The natural language toolkit. In *Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics-Volume 1*, pages 63–70. Association for Computational Linguistics, 2002.

[18] Steven Bird and Edward Loper. NLTK: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.

[19] Sameera A Abdul-Kader and John Woods. Survey on chatbot design techniques in speech conversation systems. *International Journal of Advanced Computer Science and Applications*, 6(7):72–80, 2015.

[20] Douglas Crockford. The application/JSON media type for JavaScript object notation (JSON). Technical report, 2006.

[21] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of JSON and XML data interchange formats: a case study. *Caine*, 9:157–162, 2009.

[22] Tim Bray. The JavaScript object notation (JSON) data interchange format. Technical report, 2017.

[23] Richard Wallace. The elements of AIML style. *Alice AI Foundation*, 2003.

[24] Tim Bray, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible markup language (XML) 1.0, 2008.

[25] Bayan AbuShawar and Eric Atwell. Alice chatbot: trials and outputs. *Computación y Sistemas*, 19(4):625–632, 2015.

[26] Kathleen Dahlgren and Edward Stabler. Natural language understanding system, August 11 1998. US Patent 5,794,050.

[27] Jason D Williams, Eslam Kamal, Mokhtar Ashour, Hani Amr, Jessica Miller, and Geoff Zweig. Fast and easy language understanding for dialog systems with microsoft language understanding intelligent service (LUIS). In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 159–161, 2015.

[28] Roberto Brunetti. *Windows Azure step by step*. Microsoft Press, 2011.

[29] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. Microsoft Azure and cloud computing. In *Microsoft Azure*, pages 3–26. Springer, 2015.

[30] Arijit Datta. Contextual flow in chatbot conversations, 2008.

[31] Rob High. The era of cognitive systems: An inside look at IBM Watson and how it works. *IBM Corporation, Redbooks*, 2012.