UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS, TECNOLOGIA E SAÚDE - CAMPUS ARARANGUÁ
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

Marcelo da S. Custódio

Rasa4JaCa: Uma Interface entre Sistemas Multiagentes e Tecnologias Chatbots Open Source

Araranguá
2022

Marcelo da S. Custódio

Rasa4JaCa: Uma Interface entre Sistemas Multiagentes e Tecnologias Chatbots Open Source

Trabalho de Conclusão de Curso do Curso de Graduação em Engenharia de Computação do Centro de Ciências, Tecnologia e Saúde - Campus Araranguá da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Engenharia de Computação.
Orientador: Prof. Alison R. Panisson, Dr.
Coorientadora: Profa. Débora C. Engelmann, MSc.

Araranguá
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Custódio, Marcelo da S.
Rasa4JaCa : Uma Interface entre Sistemas Multiagentes e Tecnologias Chatbots Open Source / Marcelo da S. Custódio ; orientador, Alison R. Panisson, coorientadora, Débora Cristina Engelmann, 2022.
33 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Araranguá, Graduação em Engenharia de Computação, Araranguá, 2022.

Inclui referências.

1. Engenharia de Computação. 2. Inteligência Artificial. 3. Sistemas Multiagentes. 4. Tecnologias Chatbots. I. Panisson, Alison R.. II. Engelmann, Débora Cristina. III. Universidade Federal de Santa Catarina. Graduação em Engenharia de Computação. IV. Título.

Marcelo da S. Custódio

Rasa4JaCa: Uma Interface entre Sistemas Multiagentes e Tecnologias Chatbots Open Source

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Engenharia de Computação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Computação.

Araranguá, 14 de Dezembro de 2022.

Profa. Analúcia Schiaffino Morales, Dra.
Coordenadora do Curso

Banca Examinadora:

Prof. Alison R. Panisson, Dr.
Orientador

Profa. Débora Cristina Engelmann, MSc.
Coorientadora
Instituição FACCAT e PUCRS

Prof. Alexandre Leopoldo Gonçalves, Dr.
Avaliador
Instituição UFSC

Profa. Analúcia Schiaffino Morales, Dra.
Avaliadora
Instituição UFSC

Rasa4JaCa: Uma Interface entre Sistemas Multiagentes e Tecnologias Chatbots Open Source

Marcelo da S. Custódio* Débora Cristina Engelmann†
Alison R. Panisson‡

20 de dezembro de 2022

Resumo

Sistemas Multiagentes (SMA) são uma tecnologia muito promissora no desenvolvimento de sistemas inteligentes, podendo integrar diversas técnicas de Inteligência Artificial (IA) de forma distribuída, e possibilitando a abstração de problemas complexos em crenças, desejos e intenções de agentes autônomos inteligentes, além de permitir explicabilidade através de argumentação. Entretanto a interação de usuários humanos com esses sistemas de IA geralmente acontece de forma estática, por meio de interfaces gráficas pouco amigáveis. Por outro lado, o avanço da área de Processamento de Linguagem Natural, possibilitou o desenvolvimento de Agentes Conversacionais, popularmente conhecidos como Chatbots, que são sistemas capazes de se comunicar em linguagem natural, por meio de texto ou voz, provendo uma interface amigável para os usuários do sistema. Nesse sentido, esse trabalho propõe o desenvolvimento do Rasa4JaCa, uma interface entre o Rasa, um framework de sistemas chatbots, e o JaCaMo, um framework de sistemas multiagentes. Com foco na utilização de tecnologias de código aberto. O Rasa4JaCa foi avaliado utilizando um estudo de caso já implementado em outras interfaces existentes, onde foi possível comprovar sua equivalência. Como resultado desse desenvolvimento, torna-se possível a interação em linguagem natural entre o usuário e um sistema multiagentes. Permitindo assim, a inserção de SMA no contexto de Inteligência Híbrida, em que usuários humanos e agentes inteligentes podem trabalhar em conjunto na resolução de problemas complexos.

Palavras-chaves: Inteligência Artificial. Sistemas Multiagentes. Tecnologias Chatbots.

*marcelo.custodio@grad.ufsc.br
†debora.engelmann@acad.pucrs.br
‡alison.panisson@ufsc.br

Rasa4JaCa - An Open-Source Interface between Multi-Agent Systems and Chatbots Technologies

Marcelo da S. Custódio* Débora Cristina Engelmann†
Alison R. Panisson‡

20 de dezembro de 2022

Abstract

Multi-Agent Systems (MAS) are promising technologies for intelligent systems development, they can integrate many Artificial Intelligence (AI) techniques in a distributed design, and allows the abstraction of complex systems in beliefs, desires and intentions in intelligent autonomous agents, in addition to enabling explainability through argumentation. However, the interaction between human users and these AI systems usually is given by static, and non-friendly graphic interfaces. On the other hand, the advances in Natural Language Processing, permitted the development of Conversational Agents, commonly known as Chatbots, which are systems capable of communicating, by text or speech, providing a friendly interface for the system users. In that regard, this work proposes the development of Rasa4JaCa, an interface between Rasa, a chatbot system framework, and JaCaMo, a multi-agent system framework. Focusing in using only open-source technologies. The Rasa4JaCa was evaluated with a case study already implemented using other similar technologies, proving its equivalence. As a result of our approach, natural language interaction between a user and a multi-agent system become possible. Thus allowing the insertion of MAS in the context of Hybrid Intelligence, where human users and intelligent agents can work together on complex problems solutions.

Key-words: Artificial Intelligence. Multi-Agent Systems. Chatbot Technologies.

*marcelo.custodio@grad.ufsc.br
†debora.engelmann@acad.pucrs.br
‡alison.panisson@ufsc.br

1 Introdução

Num futuro próximo, serão cada vez mais comuns as aplicações de Inteligência Artificial (IA) tomando decisões e atuando, com um grande nível de autonomia, em muitas áreas de aplicação, incluindo transporte, finanças, saúde e educação. Será necessário ter certeza que essas tecnologias estarão alinhadas com valores morais e princípios éticos humanos (DIGNUM et al., 2018). Sendo também necessário que as mesmas sejam capazes de explicar suas tomadas de decisão para seus usuários humanos, possibilitando que os mesmos entendam tais tomadas de decisão, passem a confiar nesses sistemas automatizados e também sejam capazes de gerenciá-los (GUNNING, 2017). Uma preocupação das áreas onde a tomada de decisão tem muito impacto na vida humana, tendo como exemplo a saúde e o direito, em que questões como discriminação, injustiça, privacidade das informações, opacidade e transparência dos dados, fazem toda a diferença para o usuário (EDWARDS; VEALE, 2017; ARRIETA et al., 2019). Tecnologias como Redes Neurais Artificiais, Lógica Fuzzy, Agentes Inteligentes, Sistemas Multiagentes, podem ser utilizadas para implementar sistemas de suporte a decisão, ajudando um tomador de decisões a avaliar e selecionar alternativas. Estes sistemas são particularmente úteis em problemas complexos que envolvem incertezas, grandes quantidades de dados, e não são determinísticos (PHILLIPS-WREN et al., 2009; PHILLIPS-WREN, 2012).

Sistemas baseados em agentes tem gerado muito entusiasmo nos últimos anos, por causa de sua promessa como um novo paradigma para conceitualizar, projetar, e implementar software capazes de incorporar várias técnicas de IA, como raciocínio automático e planejamento. De forma geral, a capacidade de um agente só é limitada por seu conhecimento, seus recursos de computação e suas perspectivas (YE; ZHANG; VASILAKOS, 2016). Um sistema multiagentes (SMA) é a extensão da tecnologia de agentes inteligentes. Em um sistema multiagentes, um grupo de agentes autônomos age em um ambiente para atingir um objetivo em comum, e/ou seus objetivos individuais. Esses agentes podem cooperar ou competir uns com os outros e compartilhar ou não seu conhecimento entre si (WOOLDRIDGE, 2009; BALAJI; SRINIVASAN, 2010). Também permitem a execução de diferentes tarefas de forma distribuída e tornam possível a integração com diferentes tecnologias. Esses Agentes Inteligentes são capazes de explicar suas decisões através do uso de argumentação, uma forma sofisticada de interação, habitualmente presente nos diálogos entre humanos, que pode ser utilizada de forma computacional entre agentes e traduzida em linguagem natural para o entendimento humano (PANISSON; ENGELMANN; BORDINI, 2022).

Dessa forma, SMA são tecnologias muito promissoras para o desenvolvimento de aplicações complexas, incluindo aquelas de suporte a tomada de decisão, como pode ser observado em trabalhos como (ENGELMANN et al., 2021c; PANISSON et al., 2015; SCHMIDT et al., 2016). Porém, interfaces de comunicação em linguagem natural entre agentes inteligentes e seres humanos, embora essenciais no desenvolvimento de aplicações de suporte a tomada de decisão, ainda são bastante escassas. Uma tecnologia bastante promissora para interfaces entre humano e computador são as tecnologias chatbots, que possuem grande potencial de integração com tecnologias de sistemas multiagentes. Pelo que se sabe, ainda há falta de pesquisa sobre essa potencial integração de tecnologias. A maior parte das pesquisas na área não utilizam chatbots, apenas simulações de entradas de usuário utilizando datasets (ZOLITSCHKA, 2020; GRIOL; MOLINA, 2016), ou interfaces gráficas (DUBEY et al., 2020; HERRERA.; FIGUEROA., 2018), e consequentemente, possuem um potencial menor de desenvolvimento tecnológico.

Chatbots são programas de computador desenvolvidos com técnicas de inteligência artificial que apresentam sofisticados modelos de interação entre humanos e computadores (BANSAL; KHAN, 2018). A integração de uma tecnologia chatbot em um agente autônomo inteligente, permite alcançar uma interface de comunicação com usuários em linguagem natural, permitindo ainda que o agente interaja com seus usuários humanos e outros SMA's, considerando as intenções do usuário.

Um recurso de um sistema chatbot é o conhecimento que lhe foi concedido. Usando esse conhecimento, o chatbot conduz conversas e responde questões feitas pelos usuários (NAGARHALLI; VAZE; RANA, 2020). Um de seus principais componentes é a unidade de processamento de linguagem natural, a qual permite que um chatbot seja capaz de entender comunicações em linguagem natural (sejam elas por texto ou por voz) e responder da mesma forma (KHANNA et al., 2015). Esses sistemas fornecem uma forma prática e interativa pela qual informação, mesmo complexa, pode ser repassada para os usuários de maneira eficiente (NAGARHALLI; VAZE; RANA, 2020). Consequentemente, essas tecnologias começaram a ser amplamente utilizadas em diversos segmentos, como sistemas de suporte em instituições educacionais, provendo respostas em um curto período e acessível a qualquer hora (VILLANUEVA; AGUILAR-ALONSO, 2021), assistentes de alocação de leitos para hospitais (ENGELMANN et al., 2021a), bots em sistemas de mensagem ajudando os usuários a alterarem seus hábitos alimentares não saudáveis, e dando dicas sobre exercícios físicos para que consigam mudar (FADHIL, 2018), ou um sistema para responder perguntas sobre vacinas relacionadas a COVID-19 (PELOZO; CUSTÓDIO; PANISSON, 2022), entre outros.

Neste trabalho, foi investigado o desenvolvimento de uma integração de Sistemas Multiagentes e Chatbots, tornando possível a interação entre humanos e agentes inteligentes através de linguagem natural. Para realizar a integração proposta foram utilizados os frameworks JaCaMo (BOISSIER et al., 2013), que provê uma perspectiva de programação multiagentes (Jason para programação de agentes, CArtAgO para o ambiente, e Moise para a organização dos agentes), e Rasa (RASA, 2022), um framework de aprendizado de máquina, de código livre, utilizado para automatizar conversas em texto e voz, entender mensagens, manter diálogos, e conectar a outros canais de mensagens e API's (*Application Programming Interface*). Como estudo de caso, foi utilizado um cenários de uma aplicação no domínio da saúde descrito em (ENGELMANN et al., 2021a), a qual fornece suporte á tomadas de decisão relacionadas à alocação de leitos em hospitais.

Este trabalho está estruturado em 6 seções. Nas Seções 2 e 3 encontra-se a fundamentação teórica, onde são abordadas as técnicas e tecnologias empregadas na construção de Sistemas Chatbots modernos, com foco no framework Rasa para a construção de Agentes Conversacionais Inteligentes, e tecnologias de Sistemas Multiagentes, com foco no framework JaCaMo para o desenvolvimento de Agentes Autônomos Inteligentes, respectivamente. Após, nas Seções 4 e 5, é descrito o desenvolvimento do projeto proposto e avaliação do mesmo por um estudo de caso. Na seção 6 são abordados alguns dos trabalhos relacionados. Por fim, na Seção 7 são apresentadas as conclusões e trabalhos futuros.

2 Sistemas Chatbot

Sistemas Chatbots são programas computacionais capazes de processar entradas em linguagem natural de um usuário, e responder de forma inteligente (KHAN; DAS, 2018). Esses sistemas podem ser classificados em diversos aspectos, seja por sua estrutura, seu funcionamento ou até mesmo seu domínio (LOKMAN; AMEEDEEN, 2019; BORAH et al., 2019; MCTEAR, 2021), como descrito abaixo:

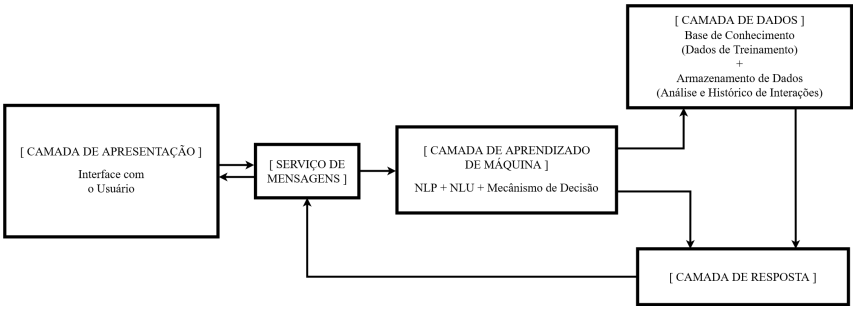
- **Geração de Resposta**: a geração da resposta de um chatbot pode ser **baseada em recuperação**, onde os usuários finais seguem um fluxo roteirizado de conversação, e as respostas são recuperadas de uma lista com base nesse fluxo, ou pode ser **baseada em geração**, onde o sistema aprende a responder inicialmente através de treinamento com uma base inicial, e posteriormente através de interação com os usuários.
- **Domínio**: sistemas chatbot podem atuar de duas formas na conversação, podendo ter conhecimento em diversos assuntos e conhecimento em geral, sendo assim considerados como **domínio aberto**, ou sendo projetados para uma aplicação ou área de conhecimento específica, como saúde, direito, finanças, tendo assim um **domínio fechado**.
- **Duração da Conversação**: com base no domínio é possível identificar se o foco do sistema será em conversações de **curta duração**, onde é fácil mapear o fluxo da conversa, ou de **longa duração** onde se torna mais difícil em se manter o contexto.
- **Contexto**: quanto a persistência do contexto na conversa, pode-se classificar em 3 categorias: (i) ***Stateless***, onde cada mensagem é tratada de forma isolada, sem manter uma memória; (ii) ***Semi Stateful***, em que é levado em consideração as últimas mensagens, podendo assim lidar com assuntos de forma distinta; e (iii) ***Stateful***, onde é mantido todo o contexto da conversa, podendo assim lidar com diferentes assuntos ao mesmo tempo.

2.1 Arquitetura de um Chatbot Inteligente

Atualmente, sistemas chatbots tem seu funcionamento em sistemas baseados em regras, ou sistemas de inteligência artificial, que interagem com usuários utilizando linguagem natural, principalmente por interfaces baseadas em texto. Esses sistemas chatbots independentes também podem ser integrados em diversas plataformas de conversação via API's, como Skype®[1], Whatsapp®[2] e Slack®[3], por exemplo (KHAN; DAS, 2018). Para um chatbot moderno e inteligente ter essas capacidades, ele deve possuir uma arquitetura robusta (BORAH et al., 2019; MCTEAR, 2021; KONG; WANG, 2021), como a representada na Figura 1.

[1] <http://www.skype.com/>
[2] <https://www.whatsapp.com/>
[3] <https://slack.com/>

Figura 1 – Arquitetura de um Chatbot Inteligente.



Fonte: Adaptado de Borah et al. (2019).

- **Camada de Apresentação**: Refere-se à interface com o usuário final, por onde esse irá interagir com o sistema, como por exemplo uma página web ou aplicativo mobile de mensagens.
- **Serviço de Mensagens**: Camada responsável pela troca de mensagens entre usuários e o sistema chatbot, gerencia conversas simultâneas.
- **Camada de Aprendizado de Máquina**: Camada responsável pelo processamento das mensagens do usuário, onde acontece o Processamento de Linguagem Natural (do inglês *Natual Language Processing*, NLP), a Compreensão de Linguagem Natural (do inglês *Natural Language Understanding*, NLU), e a decisão da próxima ação a ser tomada pelo sistema, que pode ser uma resposta ao usuário, ou a execução de um serviço para o mesmo.
- **Camada de Resposta**: Fase onde acontece a construção da resposta para o usuário final, que depende do tipo de geração de resposta para o qual o sistema foi projetado.
- **Camada de Dados**: Nessa camada encontra-se a base de conhecimento do sistema, uma grande quantidade de dados utilizada para o treinamento do chatbot, medidas de qualidade da experiência do usuário e resultados das execuções do chatbot, como detecções corretas de intenções e de entidades, por exemplo.

2.2 Processamento de Linguagem Natural com Redes Neurais Artificiais

O que possibilita o desenvolvimento de um sistema chatbot inteligente, é a utilização de diversas técnicas de IA, tanto para que o chatbot consiga interpretar as entradas do usuário, quanto para que consiga responder utilizando linguagem natural, principalmente as baseadas em aprendizado de máquina e redes neurais artificiais, apresentadas abaixo:

- ***Word Embeddings***: Representar palavras numericamente, de forma automática, para a utilização em uma rede neural artificial (RNA) é um grande desafio, dessa forma utiliza-se também RNA's para a geração de vetores densos com algumas centenas de dimensões, gerados a partir da base de dados (RUSSELL; NORVIG, 2021). Normalmente, são utilizados modelos capazes de gerar os vetores de *Word Embedding* (WE) já pré-treinados em bases de dados com muita informação, já que o treinamento de um modelo assim pode ser muito custoso computacionalmente.

- **Redes Neurais Recorrentes**: Tendo em mãos um modelo capaz de gerar WE's, é possível realizar o processamento de linguagem natural utilizando RNA's. Entretanto, linguagem natural consiste em um sequência ordenada de palavras, onde o contexto de palavras próximas é importante, neste caso uma RNA do tipo *Feed-Forward* pode não ser suficiente para tal, sendo necessário a utilização de uma Rede Neural Recorrente (do inglês *Recurrent Neural Network*, RNN), capaz de lidar com entradas sequenciais (RUSSELL; NORVIG, 2021). Em uma RNN, a informação passa por uma série de realimentações, tornando-a capaz capturar informações sobre as entradas anteriores, mantendo uma espécie de memória, útil para processamento sequencial, outra vantagem é que por ter esse processamento sequencial, as RNN's também podem obter informação sobre a influência da posição de uma entrada na sequência como um todo (MCTEAR, 2021).

- **Redes *Long-Short Term Memory***: Apesar das vantagens de uma RNN sobre uma RNA *Feed-Forward*, uma RNN tem dificuldades em aprender dependências de longo termo na sequência de entradas (OLAH, 2015). Já as redes *Long-Short Term Memory* (LSTM) utilizam dois métodos para enfrentar esse problema de contexto, um mecanismo é utilizado para a rede esquecer informações que não são mais necessárias, e um segundo mecanismo para adicionar informação à rede que pode ser necessária no futuro (MCTEAR, 2021).

- **Arquitetura *Sequence to Sequence* (Seq2Seq)**: Essa arquitetura realiza uma transformação de uma sequência de entradas em uma sequência de saídas, geralmente utilizando duas redes RNN ou LSTM em série, onde a primeira será responsável por codificar os elementos da sequência de entrada, atualizando um vetor de contexto intermediário, e a segunda em decodificar esse vetor em uma sequência saída, como por exemplo em traduções de idioma e geração de respostas em um diálogo (MCTEAR, 2021). Entretanto essa arquitetura Seq2Seq pode apresentar ainda alguns problemas, mesmo utilizando camadas LSTM, ainda é difícil manter o contexto dos elementos iniciais da sequência de entrada no vetor codificado, gerando um viés para o contexto dos últimos elementos processados, o vetor de contexto também possui um tamanho limitado, que pode não ser o suficiente para representar sentenças complexas, e aumentar esse tamanho diminui a velocidade do treinamento e pode aumentar também o *overffiting* das redes, outro problema é que por serem modelos sequenciais, a saída atual pode depender das anteriores, nesse caso o processamento paralelo se torna limitado, atrasando o treinamento das redes (RUSSELL; NORVIG, 2021).

- **Mecanismo de Atenção**: Uma maneira eficiente de resolver esses problemas é gerar um vetor de contexto para cada elemento da sequência de entradas, e utilizar esses vetores para a geração da sequência de saída, assim é possível recuperar o contexto mesmo das sequências iniciais, e o tamanho fixo do vetor de contexto não se torna um limitador (RUSSELL; NORVIG, 2021). Adicionando ainda uma camada entre as duas redes, onde são ponderados quais vetores de contexto tem maior influencia na decodificação do próximo elemento de saída, com base na última saída gerada, tem-se um mecanismo onde a rede aprende em quais entradas deve prestar mais atenção para gerar a próxima saída, selecionando somente as entradas mais importantes para a geração da saída (OLAH; CARTER, 2016; MCTEAR, 2021).

- **Arquitetura *Transformer***: Adicionando um mecanismo de atenção tanto na rede codificadora quanto na rede decodificadora, tem-se uma arquitetura *Transformer*, que possibilita que cada rede aprenda as dependências internas da sequência de entrada ou saída, respectivamente. Permitindo ao modelo capturar o contexto à longo e curto termo nas sequências e removendo sua dependência sequencial, e também possibilitando maior processamento paralelo (VASWANI et al., 2017; RUSSELL; NORVIG, 2021).

2.3 Sistemas Chatbot Orientados a Tarefas

Em sistemas chatbot de domínio fechado, muitas vezes é necessário mais do que a seleção ou geração de uma resposta para o usuário final, durante a conversa o usuário pode desejar a realização de tarefas pelo sistema, sendo assim necessário que o sistema possa realizar certas ações dentro da infraestrutura em que está funcionando, como o acesso a outros sistemas por meio de API's ou mesmo acesso direto à bases de dados para recuperar e modificar informações (MCTEAR, 2021).

Nesse caso, o sistema chatbot precisa identificar corretamente qual tarefa o usuário deseja realizar, gerando uma série de questionamentos para entender melhor quais intenções o usuário possui, e quais parâmetros e entidades estão envolvidos com a intenção do usuário. Nesse contexto, pode-se utilizar NLP baseado em RNA's durante toda a conversação entre usuário e chatbot, justamente para identificar quais as intenções e a quais entidades e parâmetros o usuário está se referindo, e então realizar a tarefa desejada com base nessas informações.

2.4 Plataformas e Frameworks de Desenvolvimento

Para simplificar e acelerar o desenvolvimento desse tipo de tecnologia, algumas empresas e comunidades de desenvolvedores criaram plataformas de criação de sistemas chatbot, como por exemplo:

- **Dialogflow®**: O Dialogflow[4] é uma plataforma de NLP, disponibilizada pela Google Cloud®, que visa facilitar o design e integração de agentes conversacionais com outros sistemas (GOOGLE, 2022). O Dialogflow possui a capacidade de reconhecer as intenções do usuário e extrair entidades nomeadas durante a conversação, que podem ser utilizadas na integração dos agentes desenvolvidos com outros sistemas por meio de requisições HTTP (*Hypertext Transfer Protocol*), que são chamadas de *Fulfillments*.

- **Watson Assistant®**: O Watson Assistant[5] é uma plataforma de desenvolvimento de assistentes conversacionais, disponibilizada pela IBM Cloud® (IBM, 2022), também pode extrair entidades e reconhecer as intenções de usuário, além de realizar chamadas HTTP para serviços externos como forma de integração com outros sistemas.

[4] <https://dialogflow.cloud.google.com/>
[5] <https://www.ibm.com/br-pt/products/watson-assistant/>

2.5 Rasa

Rasa é um framework de código aberto de Aprendizado de Máquina (AM), utilizado para o desenvolvimento de sistemas chatbots e agentes inteligentes, é modular e extensível à novas extensões e funcionalidades (KONG; WANG, 2021).

Por ser uma solução de código aberto, o Rasa possui algumas vantagens sobre outras soluções, como o custo reduzido, a possibilidade de extensão e customização de suas funções e uma documentação muito completa sobre seus componentes e funcionalidades. Segundo os autores Kong e Wang (2021), Rasa (2022), o framework Rasa consiste de quatro partes principais:

- ***NLU Pipeline***: Extração de intenções e informações de contexto do diálogo.
- ***Rasa Core***: Seleção da melhor resposta ou ação de acordo com o fluxo de diálogo.
- ***Channel and action***: Conexão do chatbot com os usuários e outros sistemas.
- ***Helper functions***: Funções auxiliares ao funcionamento do sistema.

O fluxo de dados na NLU do Rasa pode ser configurado via linguagem de marcação YAML (*YAML Ain't Markup Language*) na seção de configurações *NLU Pipeline*, como exemplificado no Listing 1. Uma configuração de fluxo de dados no Rasa geralmente possui as seguintes etapas:

- **Modelo de linguagem**: Carrega o modelo de linguagem (*Word Embedding*) responsável por representar as palavras como vetores, que serão utilizados nas próximas etapas.
- ***Tokenizer***: Responsável por separar o texto em *tokens* (símbolos).
- **Extrator de Características**: Extrai características de sequências de *tokens*, necessário tanto para a etapa de extração de entidades, quanto para a classificação de intenções.
- **Extrator de Entidades**: Realiza a extração de entidades nomeadas utilizando as características extraídas em etapas anteriores.
- **Classificador de Intenções**: Classifica o texto em diferentes intenções de usuários com base no contexto da conversa.
- **Estrutura de Saída**: Organiza a previsão dos resultados, como intenções e entidades, em uma estrutura de dados e disponibiliza para o *Rasa Core*.

Listing 1 : Exemplo de *Pipeline* da NLU.

```yaml
pipeline:
    - name: WhitespaceTokenizer
    - name: RegexFeaturizer
    - name: LexicalSyntacticFeaturizer
    - name: CountVectorsFeaturizer
    - name: CountVectorsFeaturizer
      analyzer: char_wb
      min_ngram: 1
      max_ngram: 4
    - name: DIETClassifier
      epochs: 100
      constrain_similarities: true
    - name: EntitySynonymMapper
    - name: ResponseSelector
      epochs: 100
      constrain_similarities: true
    - name: FallbackClassifier
      threshold: 0.3
      ambiguity_threshold: 0.1
```

O treinamento da NLU do Rasa é feito por meio de exemplos que devem ser disponibilizados também no formato YAML, onde deve conter uma lista de intenções possíveis que o usuário possa ter, e para cada intenção uma lista com exemplos de texto que definem essa intenção. Por exemplo, o trecho abaixo (Listing 2) demonstra os exemplos fornecidos para o reconhecimento da intenção `cumprimento`.

Listing 2 : Exemplo de Intenção.

```yaml
- intent: cumprimento
  example: |
     - Olá
     - Oi
     - Boa tarde!
     - Bom dia
     - Boa noite
     - Hey
```

Para indicar entidades nomeadas nos exemplos da NLU, essas devem ser delimitadas entre colchetes e com o nome da entidade indicado entre parenteses, caso o valor da entidade seja um sinônimo, é possível indicar o valor correto e nome da entidade no formato chave e valor utilizando JSON (*JavaScript Object Notation*). O exemplo abaixo (Listing 3) demonstra exemplos da intenção `informar_nome` com marcações da entidade `nome` fornecida nas sentenças de exemplo. Ainda é possível mapear sinônimos de palavras e adicionar novas características utilizando *Lookup Tables* e Expressões Regulares (do inglês *Regular Expressions*, RegEx), fornecendo também exemplos em YAML.

Listing 3 : Exemplo de Entidade.

```
- intent: informar_nome
  example: |
     - meu nome é [João](nome)
     - eu me chamo [Maria](nome)
     - [Juca](nome) é o meu nome
     - me chamam de [Mateus](nome)
     - Eu sou o [Thiago](nome)
```

Já no módulo *Rasa Core*, é necessário descrever o domínio do sistema chatbot utilizando também YAML, no qual são listadas todas as intenções, entidades nomeadas, ações que o sistema pode realizar, respostas pré-definidas e configurações de sessão com o usuário final. No domínio também são definidos os *Slots*, que servem como uma memória auxiliar do sistema, esses *slots* podem manter salvas informações importantes durante todo o diálogo, como intenções, entidades, valores e listas.

Além disso, é o módulo responsável por escolher a melhor resposta a ser enviada ou ação a ser realizada, baseado nas intenções do usuário e entidades extraídas durante o diálogo com o usuário, descritas no domínio do sistema. Para realizar essa seleção é necessário fornecer exemplos de diálogo também em YAML, chamados de *Stories*, onde são descritas sequências de intenções e ações ou respostas que definirão o funcionamento geral do sistema. Caso um fluxo de diálogo curto não dependa de contexto, esse pode ser descrito como uma *Rule*, onde o fluxo sempre será executado independentemente do estado da conversação. Abaixo, no Listing 4, é apresentado um exemplo de *story* simples, descrevendo um fluxo de diálogo em que após o cumprimento, o chatbot pergunta o nome do usuário, seguindo com um agradecimento.

Listing 4 : Exemplo de *Story*.

```
- story: cumprimento_padrao
  steps:
  - intent: cumprimento
  - action: utter_cumprimentar
  - action: utter_perguntar_nome
  - intent: informar_nome
  - action: action_agradecer_usuario
```

Essa seleção das melhores respostas e ações é realizada via *Policies*, que devem ser configuradas pelo usuário. As *policies* definem um *pipeline* assim como na NLU, entretanto elas analisam o fluxo do diálogo descrito pelas *Stories*. Um exemplo de configuração de *policies* pode ser encontrado no Listing 5.

Listing 5 : Exemplo de *Policies*.

```
policies:
    - name: MemoizationPolicy
    - name: RulePolicy
    - name: UnexpecTEDIntentPolicy
      max_history: 5
      epochs: 100
    - name: TEDPolicy
      max_history: 5
      epochs: 100
      constrain_similarities: true
```

O Rasa possui um conjunto de ações pré-definidas que podem ser utilizadas no seu desenvolvimento, ajudando no processamento do fluxo da conversação, mas a sua principal vantagem é a possibilidade de desenvolver ações customizadas utilizando o Rasa SDK (*Software Development Kit*) ou outra tecnologia a gosto do desenvolvedor. Essas ações customizadas podem ser utilizadas pra integrar o sistema chatbot a outros sistemas, como bancos de dados e API's. Além das ações customizadas, o Rasa pode se conectar com outras aplicações de conversação, como por meio de canais definidos pelo desenvolvedor

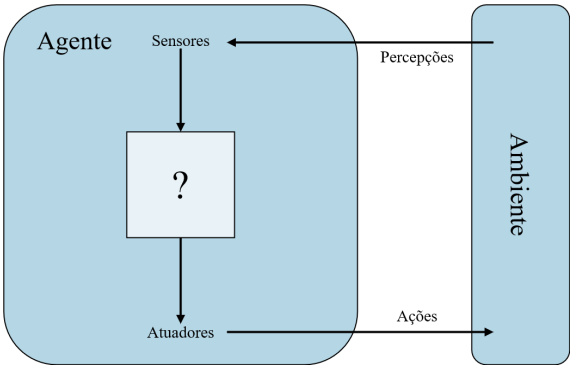
Ainda existem outras funções auxiliares no funcionamento do Rasa, que possuem versões internas funcionando, mas que podem ser customizadas e indicadas ao sistema por meio de *Endpoints* do tipo:

- ***Event broker***: Permite a conexão com outros serviços que possam processar a conversação de forma assíncrona.
- ***Tracker Store***: Todas as conversações do Rasa são armazenadas em *Tracker Stores*, que são disponibilizadas pelo framework ou podem ser customizadas.
- ***Lock Store***: Mecanismo utilizado para garantir que as mensagens para os usuários sejam sempre processadas na ordem correta.
- **Ações customizadas**: Servidor onde são executadas todas as ações customizadas.
- **Servidor de Geração de Linguagem Natural**: Alternativa externa para geração de respostas do Rasa, ao contrário da seleção de respostas padrão.
- **Servidor de Modelos**: Permite ao Rasa recuperar seus modelos treinados de um servidor externo.

3 Sistemas Multiagentes

Um agente autônomo inteligente é um sistema computacional, situado em algum ambiente (Figura 2), e é capaz de realizar ações de forma autônoma nesse ambiente, a fim de cumprir seus objetivos (WOOLDRIDGE, 2009). São sistemas reativos com um certo grau de autonomia, onde se pode delegar alguma tarefa para eles, e o próprio sistema determina a melhor maneira de realizar a tarefa, ao invés de precisarem de instruções de baixo nível para realiza-las (BORDINI; HÜBNER; WOOLDRIDGE, 2007).

Figura 2 – Agente Autônomo.



Fonte: Adaptado de Russell e Norvig (2021).

Um Sistema Multiagentes é aquele que consiste de vários agentes autônomos, interagindo entre si tipicamente trocando mensagens através de alguma rede de computadores. Os agentes em um Sistema Multiagentes representam ou agem a favor de seus usuários com diversos objetivos e motivações, e precisam ter a capacidade de interagir entre si, para poderem cooperar, competir, coordenar e negociar com outros agentes, a fim de realizarem seus objetivos (WOOLDRIDGE, 2009).

3.1 Arquitetura de um Agente

Uma das arquiteturas de agentes mais conhecida, é a arquitetura BDI (*Belief-Desire-Intention*). O modelo BDI se origina da teoria do raciocínio prático humano, uso da razão para decidir como agir, desenvolvido pelo filósofo Michael Bratman (1987), que se concentra particularmente no papel das intenções no raciocínio prático.

A principal ideia do modelo BDI é a de que se pode falar sobre programas de computador fazendo referência ao seu 'estado mental'. Isso torna possível o entendimento do comportamento de um sistema complexo através da atribuição de atitudes como 'acreditar' e 'desejar', como uma ferramenta de abstração. Isso permite prever e explicar sucintamente o comportamento de sistemas complexos sem ter que entender como eles realmente funcionam (BORDINI; HÜBNER; WOOLDRIDGE, 2007).

- **Crenças (*Beliefs*)**: São informações que o agente tem sobre o mundo, logo dependem da capacidade do agente de percebe-las e coleta-las. Essas informações podem estar desatualizadas ou serem imprecisas.
- **Desejos (*Desires*):** São todos os possíveis estados de mundo que o agente gostaria de alcançar, e representam a motivação de um agente. Ter um desejo, no entanto, não implica que um agente atue sobre ele, pois podem haver outras prioridades e um desejo não ser perseguido.
- **Intenções (*Intentions*):** São os estados dos desejos (objetivos) que o agente decidiu realizar. As intenções podem ser objetivos que são delegadas ao agente, ou podem resultar da consideração de opções para alcançar o mesmo objetivo, onde o agente analisa suas opções e escolhe uma delas.

3.2 Ciclo de Raciocínio e Planejamento

O principal modelo de decisão no modelo BDI é conhecido como Raciocínio Prático, que consiste no raciocínio realizado pelos agentes com base em suas crenças, desejos e intenções, deliberando quais ações tomar para realizar seus objetivos (BORDINI; HÜBNER; WOOLDRIDGE, 2007).

Um modelo computacional que implementa o Raciocínio Prático utilizando a arquitetura BDI é o *Procedural Reasoning System* (PRS), ou Sistema de Raciocínio Procedural, desenvolvido por Michael Georgeff e Amy Lansky (GEORGEFF; LANSKY, 1987), no Instituto de Pesquisa de Stanford.

No PRS, os agentes não criam seus próprios planos de ação, eles são equipados com uma biblioteca de planos criada pelo desenvolvedor (WOOLDRIDGE, 2009). Planos no PRS possuem as seguintes características:

- **Objetivo**: Pós-condição do plano, intenção do agente que será dada como realizada após a execução do plano.
- **Contexto**: Pré-condição do plano, crenças do agente necessárias para a realização do plano.
- **Corpo**: Lista de ações que o agente deve tomar para concluir o plano.

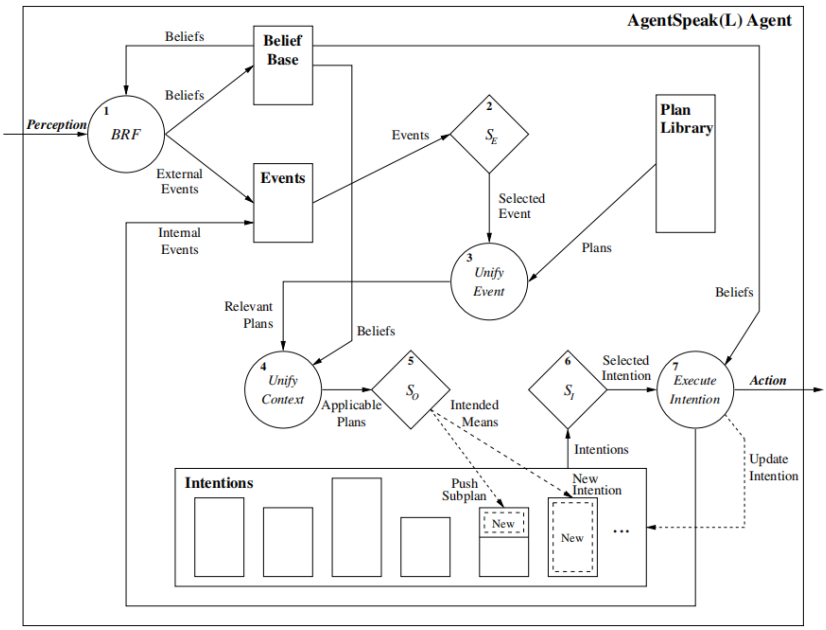
3.3 Programação Orientada a Agentes

Uma linguagem de programação orientada a agentes que implementa a arquitetura BDI e PRS de forma simples e unificada, é a AgentSpeak(L) (RAO, 1996). Rao desenvolveu uma linguagem de programação baseada em programação lógica, que provê os principais aspectos do PRS em um framework simples e uniforme, mas de forma teórica (BORDINI; HÜBNER; WOOLDRIDGE, 2007; BORDINI; HÜBNER, 2006).

Na linguagem AgentSpeak(L), a base de crenças de um agente é representada por um conjunto de predicados de primeira ordem (átomos) (BORDINI; HÜBNER, 2006), assim como em outras linguagens de programação lógica, e.g. Prolog. Um agente implementado em AgentSpeak é um sistema de planejamento reativo, reagindo a eventos relacionados a criação de objetivos, bem como mudanças na base de crenças, tanto por percepções vindas do ambiente, quanto pelas alterações realizadas na execução de planos ativados por eventos anteriores (BORDINI; HÜBNER, 2006).

Um framework de desenvolvimento de sistemas multiagentes, de forma prática, é o Jason, que implementa um interpretador de uma versão estendida do AgentSpeak (BORDINI; HÜBNER; WOOLDRIDGE, 2007). Além das capacidades do AgentSpeak citadas anteriormente, o Jason possui outras capacidades como ações internas, ações customizadas implementadas pelo desenvolvedor, e anotações de crenças e planos, por exemplo. E principalmente um sistema de comunicação de agentes baseado em *Knowledge Query and Manipulation Language* (KQML) (FININ et al., 1994), onde um agente pode interagir de diversas formas com outros agentes do sistema.

Figura 3 – Ciclo de Execução AgentSpeak.



Fonte: Bordini e Hübner (2006).

Nesse modelo do Jason (Figura 3), todo agente inicialmente terá uma biblioteca de planos, crenças iniciais sobre o ambiente, e um objetivo global (Listing 6), similar ao método 'main' em outras linguagens, como Java ou C. Ao ser executado, o objetivo global do agente é colocado em uma Pilha de Intenções, que contém durante a execução todos os objetivos que o agente deseja realizar, e o agente procura na sua biblioteca de planos quais planos tem em sua pós-condição o elemento no topo da pilha de intenções (BORDINI; HÜBNER; WOOLDRIDGE, 2007).

Listing 6 : Exemplo de Base de Crenças e Objetivo Global.

```
1  /* Initial beliefs and rules */
2  nome_usuario(jonas)
3  periodo(manha)
4
5  /* Initial goals */
6  !cumprimentar_usuario.
```

Desses planos selecionados apenas alguns terão sua pré-condição satisfeita com base nas crenças atuais do agente e se tornam opções a serem deliberadas pelo agente. O plano escolhido é então executado, podendo adicionar novos objetivos na pilha de intenções, encontrar outros planos para a execução, ou executar ações diretas (e.g. cálculos numéricos). Caso o plano venha a falhar, outro plano que atenda à pós-condição e pré-condição é selecionado da biblioteca de planos (WOOLDRIDGE, 2009). Listing 7 demonstra um exemplo de biblioteca de planos.

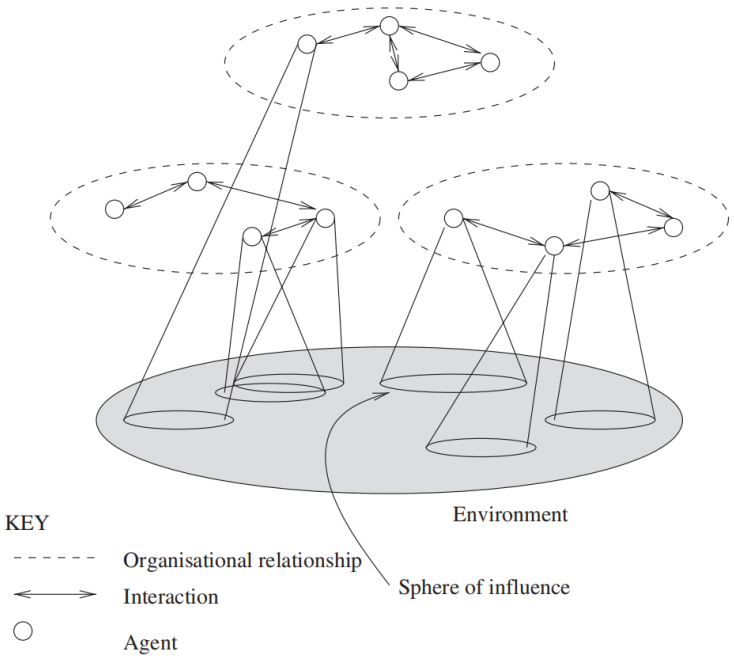
Listing 7 : Exemplo de Biblioteca de Planos.

```
1   /* Plans */
2   +!cumprimentar_usuario : nome_usuario(Nome) & periodo(manha)
3       <- !dizer_para("Bom dia!", Nome).
4
5   +!cumprimentar_usuario : nome_usuario(Nome) & periodo(tarde)
6       <- !dizer_para("Boa tarde!", Nome).
7
8   +!cumprimentar_usuario : nome_usuario(Nome) & periodo(noite)
9       <- !dizer_para("Boa noite!", Nome).
10
11  +!cumprimentar_usuario:
12      <- !dizer("Olá!").
```

3.4 Ambientes

Um Sistema Multiagentes consiste em uma população de entidades autônomas (agentes) situados em uma outra entidade estruturada (o ambiente), onde esses agentes podem executar ações para que possam alcançar seus objetivos, modificando o ambiente e/ou interagindo com outros agentes (WEYNS et al., 2005; BORDINI; HÜBNER; WOOLDRIDGE, 2007; WOOLDRIDGE; JENNINGS, 1995). O ambiente também é uma entidade ativa, com seus próprios processos que podem mudar seu estado independentemente dos agentes presentes no mesmo (WEYNS et al., 2005). A arquitetura de um SMA pode ser vista na Figura 4.

Figura 4 – Sistema Multiagentes.

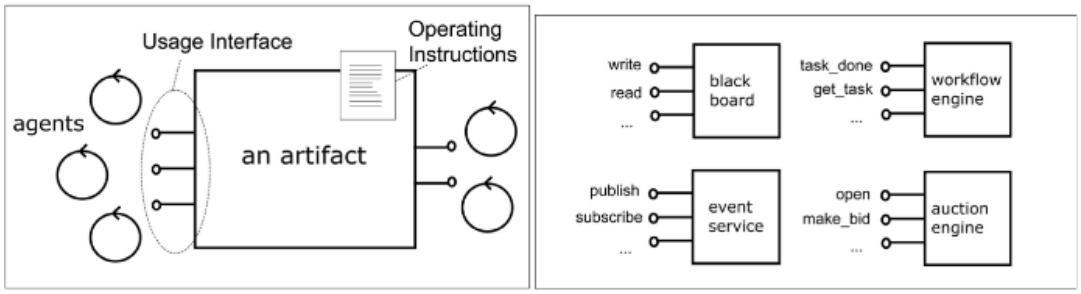


Fonte: Bordini, Hübner e Wooldridge (2007).

Geralmente, os ambientes são imaginados como um espaço físico, onde os agentes possuem um 'corpo' que pode receber percepções e interagir com esse espaço, como se fossem robôs, mas além dessa realidade, os ambientes também podem funcionar de maneira puramente lógica (WEYNS et al., 2005), sendo utilizado como uma forma robusta de memória compartilhada, habilitando os agentes a usá-lo como um meio de coordenação indireta (WOOLDRIDGE, 2009; WEYNS et al., 2005; WEYNS et al., 2005).

É possível ainda adicionar ao conceito de ambiente, objetos com os quais os agentes podem interagir, e que podem auxiliar o agente em seu objetivo, como o acesso a recursos ou como ferramentas. Esses objetos, chamados **Artefatos** (Figura 5), segundo Ricci, Viroli e Omicini (2006b) podem representar qualquer entidade que pertença ao ambiente, e fora da mente do agente, que é criada, compartilhada, usada e descartada pelos agentes, mas que não seja autônoma ou pró-ativa (ou seja, que não seja um agente).

Figura 5 – Artefatos CArtAgO.



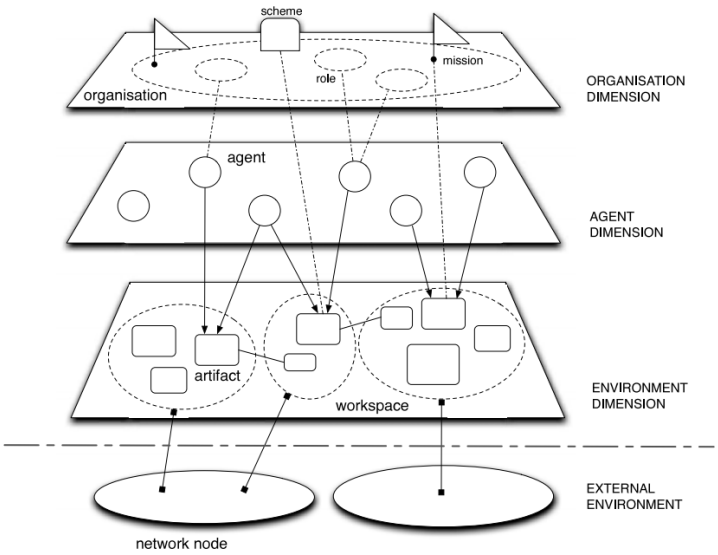
Fonte: Ricci, Viroli e Omicini (2006b).

Esses artefatos podem ser utilizados para o desenvolvimento de *Workspaces*, ambientes internos munidos de objetos e ferramentas que podem dar suporte tanto à habilidade individual quanto social dos agentes, conforme citado em (RICCI; VIROLI; OMICINI, 2007). Através desses *workspaces* é possível descrever o conceito de localidade, indicando quais artefatos um agente pode observar e utilizar.

3.5 JaCaMo

JaCaMo é um framework de Programação Multiagentes, que combina três tecnologias diferentes, (1) Moise para a organização de agentes autônomos programados em (2) Jason, atuando em ambientes distribuídos e baseados em artefatos programados em (3) CArtAgO (BOISSIER et al., 2013). Logo, o JaCaMo integra essas três plataformas definindo um link semântico entre conceitos de diferentes dimensões da programação de sistemas multiagentes, agentes, ambientes e organizações, de forma a obter um modelo de programação uniforme e consistente, como demonstrado na Figura 6.

Figura 6 – Sistema Multiagentes JaCaMo.



Fonte: Boissier et al. (2013).

4 Interface entre Rasa e JaCaMo

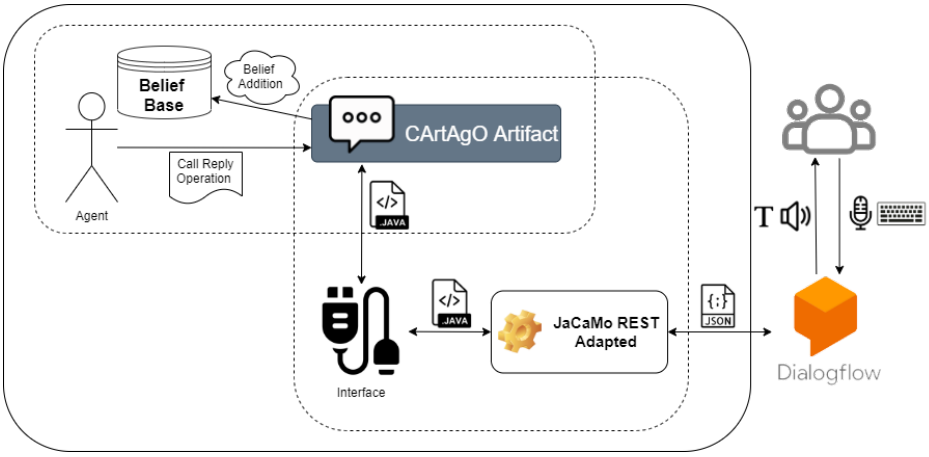
O trabalho desenvolvido foi baseado no Dial4JaCa[6] (ENGELMANN et al., 2021b; ENGELMANN et al., 2021c), uma integração entre o Dialogflow, uma plataforma de desenvolvimento de sistemas chatbot disponível na Google Cloud, e o framework de desenvolvimento de sistemas multiagentes JaCaMo (BOISSIER et al., 2020). Na seção seguinte, o Dial4JaCa será detalhado, enfatizando as características utilizadas pela abordagem proposta nesse trabalho.

4.1 Dial4JaCa

O Dial4JaCa (ENGELMANN et al., 2021a; ENGELMANN et al., 2021b) é uma ferramenta que integra o framework JaCaMo com a plataforma Dialogflow, permitindo o desenvolvimento de agentes inteligentes com a habilidade de se comunicar com seres humanos através de linguagem natural. Para tal, o Dial4JaCa é baseado no projeto de código aberto JaCaMo REST[7] (AMARAL; HÜBNER; KAMPIK, 2020), que permite com que um sistema multiagentes desenvolvido com o framework JaCaMo, possa interagir com serviços e aplicações web, e ser gerenciado e atualizado por outras aplicações. A Figura 7 demonstra uma visão geral da arquitetura do Dial4JaCa, enfatizando o uso do JaCaMo REST e artefatos CArtAgO (RICCI; PIUNTI; VIROLI, 2011; RICCI; VIROLI; OMICINI, 2006a) que deram suporte no desenvolvimento do mesmo.

[6] <https://github.com/smart-pucrs/Dial4JaCa>
[7] <https://github.com/jacamo-lang/jacamo-rest>

Figura 7 – Arquitetura do Dial4JaCa.



Fonte: Engelmann et al. (2021a).

No Dial4JaCa, quando um *fullfilment* é executado pelo Dialogflow, este é enviado no formato JSON para o JaCaMo REST, e desserializado em um objeto Java que é disponibilizado à um artefato do CArtAgO, responsável por adicionar a requisição, com todas as informações relevantes, na base de crenças dos agentes que o estão observando o artefato. Dessa forma, os agentes podem decidir se reagirão ou não à requisição. O formato da crença disponibilizada aos agentes pode ser conferido na Listing 8.

Listing 8 : Exemplo de Base de Crença gerada pelo Dial4JaCa.

```
request(RequestedBy, ResponseId, IntentName,
    [param(Key, Value), param(Key1, Value1)],
    [context(Name, LifespanCount,
        [param(Key2, Value2), param(Key3, Value3)]
    )]
)
```

Como pode ser observado na crença correspondente a requisição, os agentes possuem não somente a informação referente a intenção do usuário, mas também outros parâmetros coletados pelo Dialogflow durante a conversação. Salienta-se que a crença adicionada aos agentes com acesso ao artefato de integração possui um formato padrão, onde estão todas as informações que esse agente venha a precisar para cumprir seus objetivos, ou seja, a identificação do usuário que fez a requisição, a sua intenção, os parâmetros necessários para realização da intenção, e o contexto da conversação.

Ainda, a infraestrutura desenvolvida fornece a operação de `reply`, que possibilita ao agente responder a comunicação anterior do usuário, e outros tipos de respostas úteis ao Dialogflow, `replyWithEvent` que pode alterar o fluxo de diálogo no Dialogflow, e `replyWithContext`, que pode alterar o contexto da conversação entre o usuário e o sistema. O exemplo na Listing 9 demonstra um plano que pode ser utilizado por agentes para responder um cumprimento do usuário, ou seja, a intenção `"cumprimento"`, respondendo com `"Olá, como posso te ajudar?"`.

Listing 9 : Exemplo de Plano para tratamento da Crença gerada pelo Dial4JaCa.

```
1    +request(RequestedBy, ResponseId, IntentName, Params, Contexts):
2        (IntentName == "cumprimento")
3        <-
4            reply("Olá, como posso te ajudar?").
```

4.2 Rasa4JaCa

Rasa4JaCa[8] estende o trabalho desenvolvido pelos autores Engelmann et al. (2021b), Engelmann et al. (2021a), para possibilitar a integração da tecnologia de desenvolvimento de chatbots Rasa, com o framework de desenvolvimento de sistemas multiagentes JaCaMo.

O desenvolvimento do Rasa4JaCa visa fornecer ao Rasa, a mesma funcionalidade que o Dialogflow possui com o Dial4JaCa, um recurso poderoso de integração com sistemas multiagentes, já que o Rasa é totalmente gratuito, possui seu código aberto, e é um dos frameworks de sistemas chatbots mais utilizados da atualidade.

O Rasa4JaCa é implementado através de um artefato CArtAgO, que desserializa a requisição do Rasa em um objeto Java, e adiciona uma crença na base de crenças dos agentes que o estão observando, essa crença possui a intenção do usuário e todos os parâmetros necessários para atendê-la.

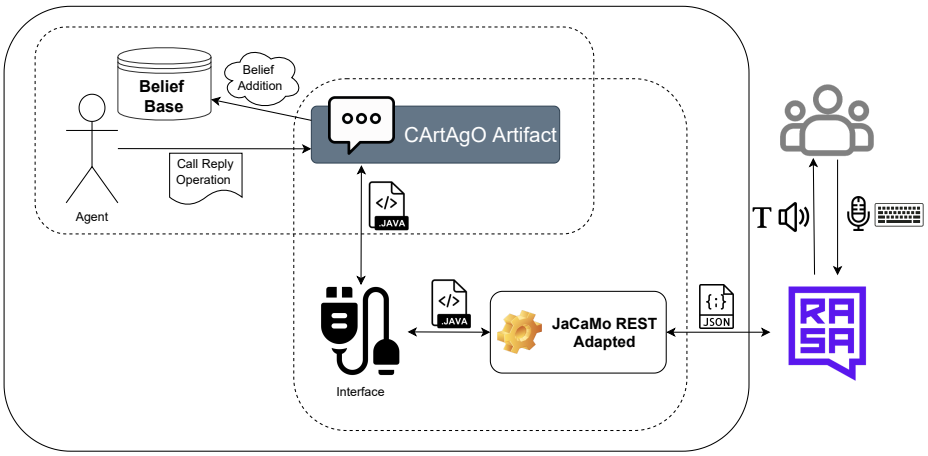
O que torna possível a extensão do Dial4JaCa para dar suporte ao Rasa, são as semelhanças entre o Rasa e o Dialogflow, a começar por ambas as plataformas serem orientadas às intenções do usuário e possuírem a capacidade de extrair entidades nomeadas da conversação com o usuário. Entretanto, enquanto o Dialogflow possui um sistema de contexto que deve ser construído pelo desenvolvedor na construção da base de dados e fluxos de diálogo, o Rasa persiste seu contexto de forma indireta com seus modelos de Aprendizado de Máquina definidos nas configurações da NLU e das *policies*, portanto não sendo algo necessário na construção da interface Rasa4JaCa.

Para a construção do Rasa4JaCa, foi necessário criar classes de objetos Java com o formato das requisições do Rasa para que o artefato CArtAgO possa tanto desserializar as requisições do usuário, quanto respondê-lo após os agentes Jason terminarem seu processamento.

A Figura 8, demonstra uma visão geral da arquitetura do Rasa4JaCa, onde basicamente é introduzida a tecnologia do Rasa como um componente modular na arquitetura do Dial4JaCa da Figura 7.

[8] <https://github.com/marcelo-custodio/Rasa4JaCa>

Figura 8 – Arquitetura do Rasa4JaCa.



Fonte: Próprio autor.

5 Estudo de Caso

Para avaliar o sistema desenvolvido, foi construído um sistema chatbot[9], utilizando o framework Rasa, em um dos domínios propostos em (ENGELMANN et al., 2021a), onde é apresentado um sistema de alocação de leitos hospitalares, e foram executados os mesmos casos de teste, disponibilizados no repositório[10] do Dial4JaCa, para demonstrar a equivalência entre ambas as interfaces. Nesses casos de teste um agente se conecta à interface proposta e atende às requisições vindas do usuário. Um exemplo do código do agente desenvolvido é apresentado no Listing 10.

Listing 10 : Planos Utilizados para o Estudo de Caso.

```
1    /* Requisição sem Parâmetros */
2    +request(RequestedBy, ResponseId, IntentName, Params, Contexts):
3        (IntentName == "Call Jason Agent")
4        <-
5            reply("Hello, I am your Jason agent, how can I help you?").
6
7    /* Requisição com Parâmetros */
8    +request(RequestedBy, ResponseId, IntentName, Params, Contexts):
9        (IntentName == "Call With Contexts and Parameters")
10       <-
11           .print("The contexts and parameters will be listed below.");
12           !printContexts(Contexts);
13           !printParameters(Params);
14           reply("Hello, I'm your Jason agent,\
15           I received your contexts and parameters").
```

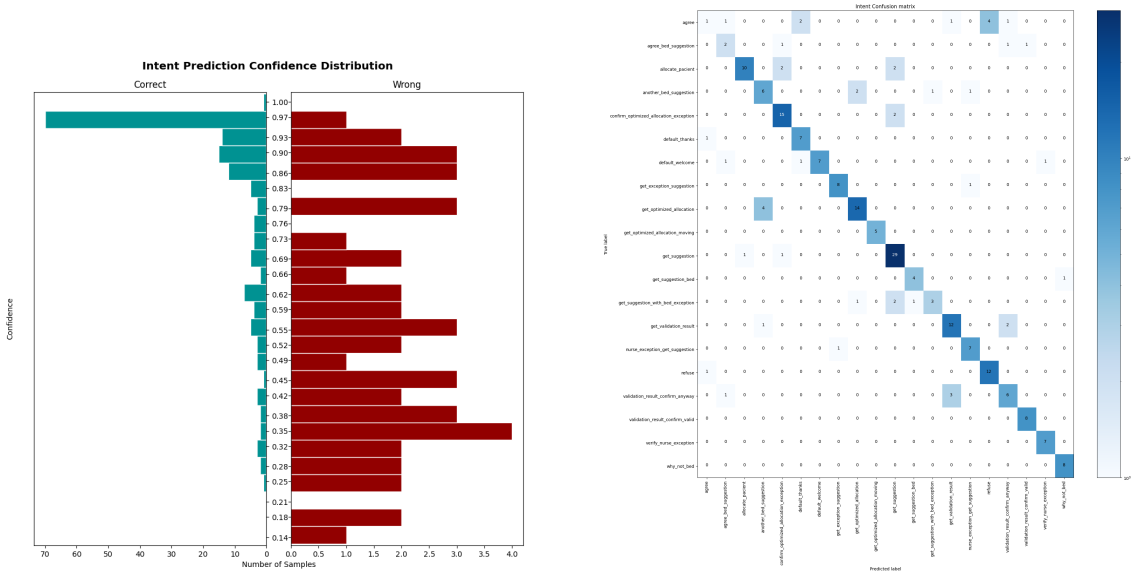
[9] <https://github.com/marcelo-custodio/rasa-bed_allocation_system>
[10] <https://github.com/smart-pucrs/Dial4JaCa>

A base de dados utilizada para o treinamento do sistema chatbot pelo Rasa foi a mesma utilizada no Dialogflow, resultando em 230 sentenças, 23 intenções e 3 entidades para o treinamento da NLU, além de 41 *stories* e 39 ações para o treinamento das *policies*. As configurações da NLU (apresentadas no Apêndice A) e *Policies* (apresentadas no Apêndice B), foram baseadas no modelo recomendado pelo Rasa para chatbots em início de desenvolvimento e com WE's pré-treinados (RASA, 2022).

Para a avaliação do sistema chatbot desenvolvido para esse estudo de caso, foram utilizadas as ferramentas de teste disponibilizadas pelo próprio framework Rasa, onde é possível avaliar as distribuições de confiança e matrizes de confusão na classificação das intenções do usuário e no reconhecimento de entidades.

Analisando os resultados obtidos na Tabela 1, é possível observar o resultado das métricas para os dados de treino, e o resultado reduzido para os dados de teste, o que pode indicar um *overfitting* do modelo da NLU quanto a predição de intenções. Ainda assim, pode-se considerar um bom resultado geral. Melhores resultados possivelmente seriam alcançados com mais dados de treinamento. Também, observando a distribuição de confiança e matriz de confusão (Figura 9) da predição de intenções, pode-se perceber que para níveis altos de confiança, a predição de intenção é feita majoritariamente de maneira correta. É importante frisar que as escalas para predições corretas e predições incorretas da Figura 9 (a) diferem.

Figura 9 – Predição de Intenções.



(a) Distribuição de Confiança.

(b) Matriz de Confusão

Fonte: Próprio autor.

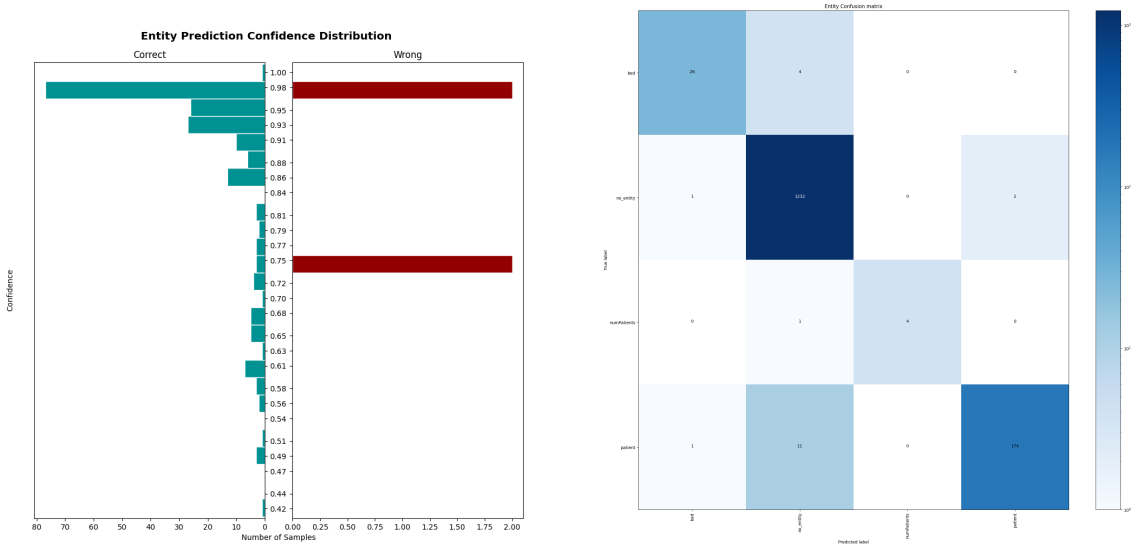
Tabela 1 – Métricas do Modelo de Predição de Intenções.

Métrica	Dados de Treino	Dados de Teste
Acurácia	0.999	0.784
F1-score	0.999	0.764
Precisão	0.999	0.787

Fonte: Próprio Autor.

Diferentemente da predição de intenção, a extração de entidades apresentou métricas com valores altos tanto nos dados de teste quanto de treino (Tabela 2), indicando uma boa generalização do modelo. Além de seus acertos terem confiança majoritariamente alta, como mostrado na Figura 10. Assim como na figura 9 (a), as escalas para predições corretas e predições incorretas na Figura 10 (a) também diferem.

Figura 10 – Extração de Entidades.



(a) Distribuição de Confiança.

(b) Matriz de Confusão.

Fonte: Próprio autor.

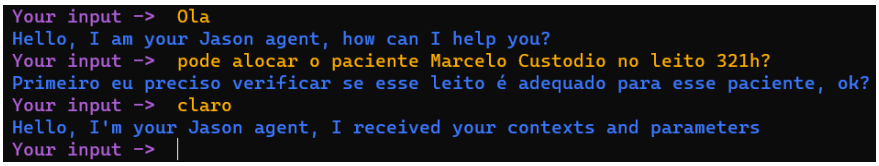
Tabela 2 – Métricas do Modelo de Extração de Entidades.

Métrica	Dados de Treino	Dados de Teste
Acurácia	0.993	0.986
F1-score	0.977	0.948
Precisão	0.992	0.977

Fonte: Próprio Autor.

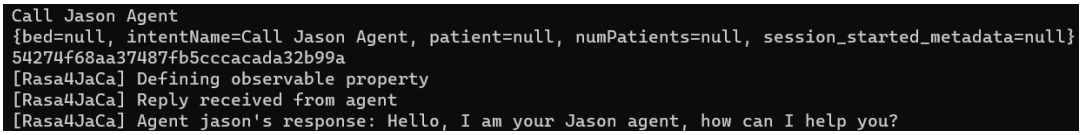
Já para a avaliação do Rasa4JaCa, os casos de teste avaliam intenções, eventos, contexto e parâmetros percebidos pelo agente através da interface. No caso do Rasa4JaCa, como os eventos e contexto não são necessários, foram avaliados somente os testes referentes as intenções e parâmetros recebidos, como mostrado no Listing 10. Os resultados podem ser observados nas Figuras 11 e 12, onde é simulada a conversação de um usuário com um agente desenvolvido no JaCaMo. Pode-se perceber que as intenções e entidades mencionadas pelo usuário foram repassadas com sucesso ao agente.

Figura 11 – Utilização do Chatbot Rasa via Terminal.

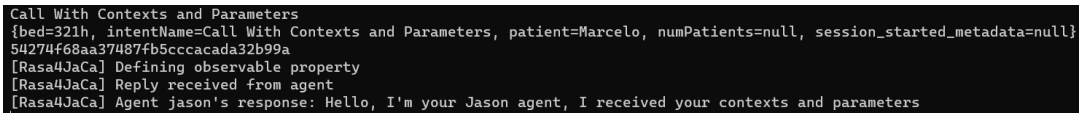


Fonte: Próprio autor.

Figura 12 – Informações Recebidas e Enviadas pelo Rasa4JaCa.



(a) Requisição sem Parâmetros.



(b) Requisição com Parâmetros.

Fonte: Próprio autor.

6 Trabalhos Relacionados

Existem alguns trabalhos relacionados ao Rasa4JaCa. Como descrito na Seção 4, este trabalho foi baseado no Dial4JaCa (ENGELMANN et al., 2021a), estendendo essa tecnologia para incorporar o Rasa. O Dial4JaCa implementa uma interface entre sistemas multiagentes desenvolvidos no framework JaCaMo e agentes conversacionais desenvolvidos na plataforma Dialogflow. Ele tem sido aplicado em vários domínios, por exemplo, para o desenvolvimento de um sistema de alocação de leitos, estendendo o trabalho (ENGELMANN et al., 2021c), um sistema de coordenação de tarefas em grupo voltado para ambientes educacionais, que estende o trabalho (COLISSI et al., 2021), entre outros.

Outro trabalho na mesma direção que o Rasa4JaCa e Dial4JaCa é apresentado em (BAYSER et al., 2018). Onde foi desenvolvido o Ravel, um SMA equipado com uma NLU e outros componentes para a orquestração de diálogos. Utilizando uma arquitetura de SMA baseada em microsserviços, em conjunto ao Watson Assistant®, o Ravel foi aplicado em um cenário de assessoria financeira, apresentando mensagens coerentes em 97% dos diálogos.

Também, em (LAEEQ; MEMON, 2021), foi desenvolvido o Scavenge, um assistente de voz baseado em SMA, para o suporte de Sistemas de Gestão de Aprendizagem, aplicado ao Moodle®[11]. O Scavenge foi desenvolvido utilizando o framework JADE (*Java Agent DEvelopment framework*), utilizou a biblioteca "Artyom.js"[12] para reconhecimento de voz, e uma biblioteca de regras em AIML (*Artificial Intelligence Markup Language*) para a realização do processamento de linguagem natural da voz. Os estudantes utilizando um sistema de gestão de aprendizagem, apoiado pelo Scavenge, demonstraram maior motivação, além de completarem mais tarefas em um período menor de tempo.

O Rasa4JaCa difere dos trabalhos (LAEEQ; MEMON, 2021) e (BAYSER et al., 2018). Sendo que: (i) o Scavenge (LAEEQ; MEMON, 2021) não utiliza uma plataforma ou framework de desenvolvimento de chatbots, mas sim um sistema baseado em regras para o processamento da linguagem natural, além de utilizar outro framework para o desenvolvimento de agentes inteligentes. (ii) o Ravel (BAYSER et al., 2018) foca no desenvolvimento de um SMA equipado com NLU e não uma interface de integração entre tecnologias chatbots e SMA's, bem como utiliza uma tecnologia diferente para o desenvolvimento de chatbots. Por último, apesar do Rasa4Jaca ser baseado no Dial4JaCa, ele utiliza o Rasa como componente de integração, que possui diversas vantagens, por ser gratuito e de código aberto.

7 Conclusão

Nesse trabalho foi proposta uma interface entre sistemas multiagentes e sistemas chatbots intitulada Rasa4JaCa. O trabalho desenvolvido é baseado no Dial4JaCa (ENGELMANN et al., 2021a; ENGELMANN et al., 2021b), estendendo essa iniciativa com a integração do framework de desenvolvimento de chatbots Rasa. Dessa forma, possibilita-se o desenvolvimento de aplicações multiagentes utilizando o framework de desenvolvimento JaCaMo, onde agentes podem se comunicar com usuários do sistema em linguagem natural.

Com a avaliação do estudo de caso realizada, foi confirmada a paridade com o sistema desenvolvido em (ENGELMANN et al., 2021a), onde um sistema chatbot interagindo com um usuário, é capaz de se comunicar com um SMA, fornecendo todas as informações necessárias para que os agentes possam realizar as ações necessárias para que possam cumprir seu propósito. Também pode-se perceber a equivalência entre o Dialogflow e o Rasa em um mesmo domínio. E, com o desenvolvimento da tecnologia descrita nesse trabalho, tem-se opções de desenvolvimento de aplicações que não venham a depender do Dialogflow, o que é bastante positivo, considerando que o Rasa além de totalmente gratuito, oferece outras vantagens por ser uma ferramenta de código aberto, e altamente customizável.

Essa abordagem possibilita o desenvolvimento de aplicações utilizando o paradigma de sistemas multiagentes no contexto de Inteligência Híbrida (AKATA et al., 2020), onde usuários humanos e agentes autônomos inteligentes podem trabalhar em conjunto para a resolução de problemas complexos, como acontece em cenários como alocação de leitos hospitalares (ENGELMANN et al., 2021c), facilitando a forma com que profissionais da saúde interagem com o sistema de alocação, e ao mesmo tempo oferecendo o controle e decisão final, cruciais na aplicação, juntamente com o suporte provido pelos agentes inteligentes.

[11] <https://moodle.com/pt/>
[12] <https://sdkcarlos.github.io/sites/artyom.html>

Também, o trabalho desenvolvido abre portas para a integração de outras técnicas de IA, já que cada agente autônomo inteligente pode possuir uma funcionalidade diferente dentro do SMA, com seus próprios conhecimentos sobre o mundo e habilidades específicas para cumprir seus objetivos, tal como a capacidade de aprender com *Reinforcement Learning* ou perceber características visuais do mundo real utilizando Visão Computacional.

Nesse sentido, a extensão deste trabalho pode se dar como descrito acima, integrando outras técnicas de IA em um sistema capaz de interagir com seus usuários em linguagem natural, estendendo assim a capacidade dos agentes presentes no SMA por meio de novas percepções e capacidades, e possibilitando a inserção de um sistema autônomo ou de suporte à decisão em outros domínios. Assim como utilizar a capacidade de argumentação lógica em linguagem natural, pertencente aos agentes com arquitetura BDI, descrita por Panisson, Engelmann e Bordini (2022), possibilitando que o sistema possa argumentar com o usuário até que ambos cheguem a um consenso. Pretende-se ainda utilizar o Rasa4JaCa no desenvolvimento de aplicações de sistemas multiagentes no contexto de configuração de simulações baseadas em sistemas multiagentes, estendendo o trabalho desenvolvido em (CUSTÓDIO et al., 2022).

Agradecimentos

Esse trabalho foi financiado pelo CNPq.

Referências

AKATA, Z. et al. A research agenda for hybrid intelligence: augmenting human intellect with collaborative, adaptive, responsible, and explainable artificial intelligence. *Computer*, IEEE Computer Society, v. 53, n. 08, p. 18–28, 2020. Citado na página [25].

AMARAL, C. J.; HÜBNER, J. F.; KAMPIK, T. Towards jacamo-rest: a resource-oriented abstraction for managing multi-agent systems. *arXiv preprint arXiv:2006.05619*, 2020. Citado na página [18].

ARRIETA, A. B. et al. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *CoRR*, abs/1910.10045, 2019. Disponível em: <http://arxiv.org/abs/1910.10045>. Citado na página [3].

BALAJI, P. G.; SRINIVASAN, D. An introduction to multi-agent systems. In: *Innovations in multi-agent systems and applications-1*. [S.l.]: Springer, 2010. p. 1–27. Citado na página [3].

BANSAL, H.; KHAN, R. A review paper on human computer interaction. *International Journals of Advanced Research in Computer Science and Software Engineering*, v. 8, p. 53–56, 2018. Citado na página [4].

BAYSER, M. Gatti de et al. Ravel: a mas orchestration platform for human-chatbots conversation. In: . [S.l.: s.n.], 2018. Citado (2) vezes nas páginas [24 e 25].

BOISSIER, O. et al. *Multi-agent oriented programming: programming multi-agent systems using JaCaMo*. [S.l.]: MIT Press, 2020. Citado na página [18].

BOISSIER, O. et al. Multi-agent oriented programming with jacamo. *Science of Computer Programming*, v. 78, n. 6, p. 747–761, 2013. ISSN 0167-6423. Special section: The Programming Languages track at the 26th ACM Symposium on Applied Computing (SAC 2011) & Special section on Agent-oriented Design Methods and Programming Techniques for Distributed Computing in Dynamic and Complex Environments. Disponível em: <https://www.sciencedirect.com/science/article/pii/S016764231100181X>. Citado (3) vezes nas páginas [4, 17 e 18].

BORAH, B. et al. Survey of textbased chatbot in perspective of recent technologies. In: MANDAL, J. K. et al. (Ed.). *Computational Intelligence, Communications, and Business Analytics*. Singapore: Springer Singapore, 2019. p. 84–96. ISBN 978-981-13-8581-0. Citado (2) vezes nas páginas [5 e 6].

BORDINI, R. H.; HÜBNER, J. F. Bdi agent programming in agentspeak using jason. In: TONI, F.; TORRONI, P. (Ed.). *Computational Logic in Multi-Agent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 143–164. ISBN 978-3-540-33997-7. Citado (2) vezes nas páginas [14 e 15].

BORDINI, R. H.; HÜBNER, J. F.; WOOLDRIDGE, M. *Programming Multi-Agent Systems in AgentSpeak Using Jason (Wiley Series in Agent Technology)*. Chichester, West Sussex, England: John Wiley & Sons, Inc., 2007. ISBN 978-0-470-02900-8. Citado (5) vezes nas páginas [12, 13, 14, 15 e 16].

BRATMAN, M. *Intention, Plans, and Practical Reason*. [S.l.]: Cambridge: Cambridge, MA: Harvard University Press, 1987. Citado na página [13].

COLISSI, M. da S. et al. A chatbot that uses a multi-agent organization to support collaborative learning. In: STEPHANIDIS, C.; ANTONA, M.; NTOA, S. (Ed.). *HCI International 2021 - Posters*. Cham: Springer International Publishing, 2021. p. 31–38. ISBN 978-3-030-78645-8. Citado na página [24].

CUSTÓDIO, M. et al. Um sistema de simulação de epidemias utilizando sistemas multiagentes. Não publicado. 2022. Citado na página [26].

DIGNUM, V. et al. Ethics by design: Necessity or curse? In: *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. New York, NY, USA: Association for Computing Machinery, 2018. (AIES '18), p. 60–66. ISBN 9781450360128. Disponível em: <https://doi.org/10.1145/3278721.3278745>. Citado na página [3].

DUBEY, A. et al. Haco: A framework for developing human-ai teaming. In: *Proceedings of the 13th Innovations in Software Engineering Conference on Formerly Known as India Software Engineering Conference*. New York, NY, USA: Association for Computing Machinery, 2020. (ISEC 2020). ISBN 9781450375948. Disponível em: <https://doi.org/10.1145/3385032.3385044>. Citado na página [3].

EDWARDS, L.; VEALE, M. Slave to the algorithm? why a 'right to an explanation' is probably not the remedy you are looking for. *Duke Law and Technology Review*, Duke University School of Law, v. 16, n. 1, p. 1–65, dez. 2017. ISSN 2328-9600. Citado na página [3].

ENGELMANN, D. et al. Dial4jaca–a communication interface between multi-agent systems and chatbots. In: SPRINGER. *International conference on practical applications of agents and multi-agent systems*. [S.l.], 2021. p. 77–88. Citado (7) vezes nas páginas [4, 18, 19, 20, 21, 24 e 25].

ENGELMANN, D. et al. Dial4jaca–a demonstration. In: SPRINGER. *International Conference on Practical Applications of Agents and Multi-Agent Systems*. [S.l.], 2021. p. 346–350. Citado (3) vezes nas páginas [18, 20 e 25].

ENGELMANN, D. C. et al. A conversational agent to support hospital bed allocation. In: SPRINGER. *Brazilian Conference on Intelligent Systems*. [S.l.], 2021. p. 3–17. Citado (4) vezes nas páginas [3, 18, 24 e 25].

FADHIL, A. *Can a Chatbot Determine My Diet?: Addressing Challenges of Chatbot Application for Meal Recommendation*. arXiv, 2018. Disponível em: <https://arxiv.org/abs/1802.09100>. Citado na página [4].

FININ, T. et al. Kqml as an agent communication language. In: *Proceedings of the Third International Conference on Information and Knowledge Management*. New York, NY, USA: Association for Computing Machinery, 1994. (CIKM '94), p. 456–463. ISBN 0897916743. Disponível em: <https://doi.org/10.1145/191246.191322>. Citado na página [14].

GEORGEFF, M. P.; LANSKY, A. L. Reactive reasoning and planning. In: *AAAI*. [S.l.: s.n.], 1987. v. 87, p. 677–682. Citado na página [14].

GOOGLE. *Documentação do Dialogflow*. 2022. Disponível em: <https://cloud.google.com/dialogflow/docs>. Acesso em: 6 dez. 2022. Citado na página [8].

GRIOL, D.; MOLINA, J. M. A multiagent-based technique for dialog management in conversational interfaces. In: DEMAZEAU, Y. et al. (Ed.). *Advances in Practical Applications of Scalable Multi-agent Systems. The PAAMS Collection*. Cham: Springer International Publishing, 2016. p. 121–132. ISBN 978-3-319-39324-7. Citado na página [3].

GUNNING, D. Explainable artificial intelligence (xai). *Defense advanced research projects agency (DARPA), nd Web*, v. 2, n. 2, p. 1, 2017. Citado na página [3].

HERRERA., J. L. L.; FIGUEROA., H. V. R. Jaca-mm: A user-centric bdi multiagent communication framework applied for negotiating and scheduling multi-participant events - a jason/cartago extension framework for diary scheduling events permitting a hybrid combination of multimodal devices based on a microservices architecture. In: INSTICC. *Proceedings of the 10th International Conference on Agents and Artificial Intelligence - Volume 1: HAMT,*. [S.l.]: SciTePress, 2018. p. 318–330. ISBN 978-989-758-275-2. ISSN 2184-433X. Citado na página [3].

IBM. *Welcome to the new Watson Assistant*. 2022. Disponível em: <https://cloud.ibm.com/docs/watson-assistant>. Acesso em: 6 dez. 2022. Citado na página [8].

KHAN, R.; DAS, A. Introduction to chatbots. In: _____. *Build Better Chatbots: A Complete Guide to Getting Started with Chatbots*. Berkeley, CA: Apress, 2018. p. 1–11. ISBN 978-1-4842-3111-1. Disponível em: <https://doi.org/10.1007/978-1-4842-3111-1_1>. Citado na página [5].

KHANNA, A. et al. A study of today's ai through chatbots and rediscovery of machine intelligence. *International Journal of u-and e-Service, Science and Technology*, v. 8, n. 7, p. 277–284, 2015. Citado na página [4].

KONG, X.; WANG, G. *Conversational AI with Rasa: Build, Test, and Deploy AI-powered, Enterprise-grade Virtual Assistants and Chatbots*. Packt Publishing, 2021. ISBN 9781801077057. Disponível em: <https://www.packtpub.com/product/conversational-ai-with-rasa/9781801077057>. Citado (2) vezes nas páginas [5 e 9].

LAEEQ, K.; MEMON, Z. A. Scavenge: an intelligent multi-agent based voice-enabled virtual assistant for lms. *Interactive Learning Environments*, Routledge, v. 29, n. 6, p. 954–972, 2021. Disponível em: <https://doi.org/10.1080/10494820.2019.1614634>. Citado na página [25].

LOKMAN, A. S.; AMEEDEEN, M. A. Modern chatbot systems: A technical review. In: ARAI, K.; BHATIA, R.; KAPOOR, S. (Ed.). *Proceedings of the Future Technologies Conference (FTC) 2018*. Cham: Springer International Publishing, 2019. p. 1012–1023. ISBN 978-3-030-02683-7. Citado na página [5].

MCTEAR, M. *Conversational AI: Dialogue Systems, Conversational Agents, and Chatbots*. Morgan & Claypool Publishers, 2021. (Synthesis Lectures on Human Language Technologies). ISBN 9783031021763. Disponível em: <https://link.springer.com/book/10.1007/978-3-031-02176-3>. Citado (3) vezes nas páginas [5, 7 e 8].

NAGARHALLI, T. P.; VAZE, V.; RANA, N. A review of current trends in the development of chatbot systems. In: IEEE. *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*. [S.l.], 2020. p. 706–710. Citado na página [4].

OLAH, C. *Understanding LSTM Networks*. 2015. Disponível em: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. Acesso em: 21 nov. 2022. Citado na página [7].

OLAH, C.; CARTER, S. Attention and augmented recurrent neural networks. *Distill*, 2016. Disponível em: <http://distill.pub/2016/augmented-rnns>. Citado na página [7].

PANISSON, A. R.; ENGELMANN, D. C.; BORDINI, R. H. Engineering explainable agents: An argumentation-based approach. In: ALECHINA, N.; BALDONI, M.; LOGAN, B. (Ed.). *Engineering Multi-Agent Systems*. Cham: Springer International Publishing, 2022. p. 273–291. ISBN 978-3-030-97457-2. Citado (2) vezes nas páginas [3 e 26].

PANISSON, A. R. et al. Arguing about task reallocation using ontological information in multi-agent systems. In: *12th International Workshop on Argumentation in Multiagent Systems*. [S.l.: s.n.], 2015. v. 108. Citado na página [3].

PELOZO, M. L.; CUSTÓDIO, M.; PANISSON, A. R. Answering questions about covid-19 vaccines using chatbot technologies. In: SPRINGER. *Brazilian Conference on Intelligent Systems*. [S.l.], 2022. p. 458–472. Citado na página [4].

PHILLIPS-WREN, G. Ai tools in decision making support systems: A review. *International Journal on Artificial Intelligence Tools*, v. 21, n. 02, p. 1240005, 2012. Disponível em: <https://doi.org/10.1142/S0218213012400052>. Citado na página [3].

PHILLIPS-WREN, G. et al. An integrative evaluation framework for intelligent decision support systems. *European Journal of Operational Research*, v. 195, n. 3, p. 642–652, 2009. ISSN 0377-2217. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377221707010776>. Citado na página [3].

RAO, A. S. Agentspeak(l): Bdi agents speak out in a logical computable language. In: VELDE, W. Van de; PERRAM, J. W. (Ed.). *Agents Breaking Away*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996. p. 42–55. ISBN 978-3-540-49621-2. Citado na página [14].

RASA. *Introduction to rasa open source*. 2022. Disponível em: <https://rasa.com/docs/rasa/>. Acesso em: 21 nov. 2022. Citado (3) vezes nas páginas [4, 9 e 22].

RICCI, A.; PIUNTI, M.; VIROLI, M. Environment programming in multi-agent systems: an artifact-based perspective. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 23, n. 2, p. 158–192, 2011. Citado na página [18].

RICCI, A.; VIROLI, M.; OMICINI, A. Cartago: An infrastructure for engineering computational environments in mas. *Weyns et al.[31]. To appear*, 2006. Citado na página [18].

RICCI, A.; VIROLI, M.; OMICINI, A. Programming mas with artifacts. In: BORDINI, R. H. et al. (Ed.). *Programming Multi-Agent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 206–221. ISBN 978-3-540-32617-5. Citado na página [17].

RICCI, A.; VIROLI, M.; OMICINI, A. Give agents their artifacts: The a&a approach for engineering working environments in mas. In: *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*. New York, NY, USA: Association for Computing Machinery, 2007. (AAMAS '07). ISBN 9788190426275. Disponível em: <https://doi.org/10.1145/1329125.1329308>. Citado na página [17].

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach, Global Edition*. Pearson Education, 2021. ISBN 9781292401171. Disponível em: <http://aima.cs.berkeley.edu/global-index.html>. Citado (4) vezes nas páginas [6, 7, 8 e 13].

SCHMIDT, D. et al. An ontology-based mobile application for task managing in collaborative groups. In: *The Twenty-Ninth International Flairs Conference*. [S.l.: s.n.], 2016. Citado na página [3].

VASWANI, A. et al. *Attention Is All You Need*. arXiv, 2017. Disponível em: <https://arxiv.org/abs/1706.03762>. Citado na página [8].

VILLANUEVA, D. P. P.; AGUILAR-ALONSO, I. A chatbot as a support system for educational institutions. In: IEEE. *2021 62nd International Scientific Conference on Information Technology and Management Science of Riga Technical University (ITMS)*. [S.l.], 2021. p. 1–6. Citado na página [4].

WEYNS, D. et al. Environments for multiagent systems state-of-the-art and research challenges. In: WEYNS, D.; PARUNAK, H. V. D.; MICHEL, F. (Ed.). *Environments for Multi-Agent Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. p. 1–47. ISBN 978-3-540-32259-7. Citado (2) vezes nas páginas [16 e 17].

WEYNS, D. et al. Environments in multiagent systems. *The Knowledge Engineering Review*, Cambridge University Press, v. 20, n. 2, p. 127–141, 2005. Citado (2) vezes nas páginas [16 e 17].

WOOLDRIDGE, M. *An introduction to multiagent systems*. [S.l.]: John wiley & sons, 2009. Citado (6) vezes nas páginas [3, 12, 13, 14, 15 e 17].

WOOLDRIDGE, M.; JENNINGS, N. R. Intelligent agents: theory and practice. *The Knowledge Engineering Review*, Cambridge University Press, v. 10, n. 2, p. 115–152, 1995. Citado na página [16].

YE, D.; ZHANG, M.; VASILAKOS, A. V. A survey of self-organization mechanisms in multiagent systems. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, IEEE, v. 47, n. 3, p. 441–461, 2016. Citado na página [3].

ZOLITSCHKA, J. F. A novel multi-agent-based chatbot approach to orchestrate conversational assistants. In: ABRAMOWICZ, W.; KLEIN, G. (Ed.). *Business Information Systems*. Cham: Springer International Publishing, 2020. p. 103–117. ISBN 978-3-030-53337-3. Citado na página [3].

APÊNDICE A – Configuração da NLU utilizada no Estudo de Caso.

```yaml
pipeline:
  - name: SpacyNLP
    model: "pt_core_news_lg"
    case_sensitive: False
  - name: SpacyTokenizer
    "intent_tokenization_flag": False
    "intent_split_symbol": "_"
    "token_pattern": None
  - name: SpacyFeaturizer
    "pooling": "mean"
  - name: RegexFeaturizer
    "case_sensitive": False
    "use_word_boundaries": True
  - name: LexicalSyntacticFeaturizer
    "features": [
      ["low", "title", "upper"],
      ["BOS", "EOS", "low", "upper", "title", "digit"],
      ["low", "title", "upper"],
    ]
  - name: CountVectorsFeaturizer
    "analyzer": "word"
    "min_ngram": 1
    "max_ngram": 1
    "use_shared_vocab": False
  - name: CountVectorsFeaturizer
    analyzer: "char_wb"
    min_ngram: 1
    max_ngram: 4
  - name: DIETClassifier
    epochs: 100
    text: [256, 128]
    label: [256, 128]
    constrain_similarities: True
    use_gpu: True
  - name: EntitySynonymMapper
  - name: ResponseSelector
    epochs: 100
    hidden_layers_sizes: {'text': [], 'label': []}
    number_of_transformer_layers: 2
    transformer_size: 256
    connection_density: 0.2
    constrain_similarities: True
    model_confidence: softmax
    use_gpu: True
```

APÊNDICE B – Configuração das Policies utilizadas no Estudo de Caso.

```yaml
policies:
  - name: MemoizationPolicy
  - name: RulePolicy
  - name: UnexpecTEDIntentPolicy
    max_history: 5
    epochs: 100
  - name: TEDPolicy
    max_history: 5
    epochs: 100
    constrain_similarities: true
    connection_density: 0.2
    model_confidence: softmax
    use_gpu: True
```