Arisa Nest – Uma Plataforma Baseada na Nuvem para Desenvolvimento de Assistentes Virtuais

Saulo Popov Zambiasi[1], Ricardo J. Rabelo[2]

[1]Centro Tecnológico - Universidade do Sul de Santa Catarina (UNISUL)
Av. Pedra Branca, 25 - Cidade Universitária Pedra Branca - Palhoça - SC - Brasil

[2]Centro Tecnológico - Departamento de Automação e Sistemas
Universidade Federal de Santa Catarina (UFSC) - Florianópolis - SC - Brasil

saulopz@gmail.com, ricardo.rabelo@ufsc.br

Abstract. *The use of virtual assistants has become popularized by their adhesion by large companies. This is because they have brought benefits to a certain extent in people's daily activities or in their tasks in organizations, with reminders, automation of repetitive tasks, proposing solutions for various situations, etc.. To meet this demand, several tools have been developed to create this type of agent. Following the same trends, this paper presents the Arisa Nest platform, under the Platform as a Service model, as a tool to create virtual assistants with resources to manage information about users, create algorithms to give more effective answers, consume web services to the interoperability with other systems and with proactivity behaviors. Finally, some cases are presented using the platform in real scenarios and tests.*

Resumo. O uso dos assistentes virtuais tem se popularizado pela sua adesão por grandes companhias. Isso porque eles têm trazido benefícios, até certo ponto, nas atividades diárias das pessoas ou em suas tarefas nas organizações, com lembretes, automatização de tarefas repetitivas, propondo soluções para várias situações, etc.. Para comportar essa demanda, diversas ferramentas têm sido desenvolvidas para a criação desse tipo de agente. Seguindo o mesmo viés, esse artigo apresenta a plataforma Arisa Nest, sob o modelo de *Platform as a Service*, como uma ferramenta de criação de assistentes virtuais com recursos de gerir informações sobre os usuários, criar algoritmos para dar respostas mais eficazes, consumir serviços web para a interoperabilidade com outros sistemas e com comportamentos de proatividade. Ao final, são apresentados alguns casos utilizando a plataforma em cenários reais e de testes.

1 Introdução

Princípios como interoperabilidade, modularidade, virtualização, informações em tempo real, orientação a serviços, descentralização e autonomia têm sido considerados de grande importância para as empresas. A utilização de robôs de software (*softbots*, *bots* ou assistentes virtuais) é vista com solução para automatizar e auxiliar as pessoas na execução de algumas tarefas, em vários níveis de inteligência e autonomia [Rabelo, Romero e Zambiasi 2018]. Um tipo de *softbot*, na forma de um Software Assistente Pessoal, foi concebido em Zambiasi e Rabelo [2012] para interagir com máquinas, computadores, bancos de dados e outros sistemas de informação para auxiliar as pessoas

na execução de diversas tarefas. O bot interagia por conversa em linguagem natural via *instant messaging* no dispositivo móvel do usuário, podia mandar relatórios por e-mail e executava tarefas de forma autônoma, com ou sem a necessidade da confirmação do usuário.

O uso de assistentes virtuais para uma melhor interação com usuários na execução de variadas tarefas não é um assunto novo. Com os avanços das tecnologias de informação, algumas grandes empresas de software passaram a desenvolver sistemas computacionais assistentes para que empresas pudessem adotar seu uso nos seus processos de negócio [Markoff 2008]. Contudo, apesar desses esforços, ainda existem diversas limitações quanto à complexidade na implementação de chatbots e assistentes virtuais mais sofisticados e inteligentes, na integração com outros sistemas, flexibilidade e escalabilidade em termos de modificação e adição de novas ações, etc.

No intuito de contribuir para as pesquisas nessa linha, este artigo apresenta a plataforma Arisa Nest, baseada na nuvem, modelo *Platform as a Service* (PaaS), que permite a criação de assistentes virtuais com recursos como: editor de scripts em linguagem de programação Lua para responder ao usuário com mensagens mais eficazes; chamada a serviços web externos à plataforma; gerenciamento de crenças com base nos perfis dos usuários; editor e executor de comportamentos para pró-atividade; e interação por voz. Um conjunto de casos foram usados para testar a plataforma.

2 Assistentes Virtuais

Os Assistentes Virtuais provém da tecnologia dos *chatbots*, agentes de conversação para interação com os usuários em linguagem natural utilizando técnicas de inteligência artificial. Hoje, muitos desses artefatos de software não apenas interagem com o usuário via conversa, mas também podem executar tarefas para eles. O primeiro chatbot foi criado por Joseph Weizenbaum em 1966 com o nome de ELIZA e simulava a conversa com uma psicoterapeuta [Weizenbaum 1966]. Para Abu Shawar e Atwell [2015], um esforço bastante relevante hoje na área dos agentes de software de conversação, ou *chatbots*, é o ALICE, uma evolução do ELIZA, que utiliza uma linguagem chamada AIML (*Artificial Intelligence Markup Language*) sob XML (*eXtensible Markup Language*) para a criação da sua base de conversação, atualmente na versão 2.

Além de linguagens específicas e técnicas de programação de *chatbots*, existem também plataformas para facilitar a criação e servir como ambiente de execução, dando suporte ao ciclo de vida do bot e ferramentas de administração, contabilização, etc. Atualmente existem algumas plataformas bastante em ênfase, tais como a **Alexa** da Amazon. Todo o domínio da Alexa envolve um conjunto de dispositivos e ambiente de desenvolvimento. O usuário pode interagir com vários dispositivos. A interação com a Alexa pode se dar via software multiplataforma, além de dispositivos próprios para interação via voz, chamados de Alexa Echo. O Alexa Skills fornece um conjunto de ações e interações com a Alexa. Essas ações são as skills, que executam tarefas como tocar uma música, responder questões, fornecer informações do tempo, controlar as luzes da casa, etc.. Mas também é possível ao usuário criar seus próprios skills personalizados utilizando o Alexa Skills Kit. Ele permite a chamada de Amazon Web Services (AWS), integração com outros serviços desenvolvidos por outros em qualquer outra linguagem e em outros servidores. O Alexa Skills Kit fornece também um conjunto de APIs para integração com diversos dispositivos [ALEXA 2019]. É possível

programar serviços para serem utilizados nos Skills da Alexa através da interface da *Amazon Web Services* (AWS). Quando um skill é criado, dá-se um nome e, ao conversar com a Alexa, o usuário chama diretamente aquele skill. Os skills são responsivos, executando algo requisitado pelo usuário. Há uma plataforma para pesquisa de skills já publicados que o usuário pode agregar ao seu assistente. Em tempo, por meio dos skills, é possível construir uma sequência de diálogos entre o usuário e a Alexa [Tingiris 2018]. No AWS Lambda, o usuário paga apenas pelo tempo de computação consumido, sendo permitido uma quantidade de requisições/execuções gratuitas por mês. É possível criar, nas skills, respostas aos usuários por meio de uma estrutura de fatos em Node.js e em diversas linguagens de programação, como javascript, python, etc. [AWS 2019].

Outro exemplo é o **Watson Conversation**. Ele se caracteriza como "uma API para desenvolvimento de bots, com uma interface simples" que permite que pessoas sem conhecimentos de programação possam alimentar sua base de conhecimento. É preciso ter uma conta na IBM Cloud e, até certo ponto, pode ser utilizado de forma gratuita. A plataforma possui um ambiente de desenvolvimento da base de conversação baseada em intenções, entidades e diálogos. As intenções são ações atreladas às perguntas feitas pelo usuário. É necessário dar exemplos de frases de entrada que, posteriormente, o sistema generaliza para tentar identificar padrões nas intenções do usuário. As entidades são complementos. Por exemplo, se o usuário quer fazer um pedido de pizza, isso é uma intenção, as entidades são informações como sabor, tipo de massa, CEP da entrega, etc. Para as entidades, é possível trabalhar com sinônimos, como no caso de mussarela e queijo, ou adicionar expressões regulares. Os diálogos são utilizados para construir a interação com os usuários. Cada nó do diálogo pode responder algo, e um nó pode possuir nós filhos, ou um complemento de resposta. Os nós possuem uma interação if-then-else e podem armazenar variáveis [Mazon 2018]. As vantagens da utilização dele é a facilidade de criação, sem necessidade de conhecimentos de programação, além da generalização dos exemplos de frases de entrada do usuário nas inteções.

Outra plataforma é o **Dialogflow** da Google, que permite a criação de bots por voz e textos com tecnologias de Inteligência Artificial, com interação via site web, dispositivos móveis, Google Assistente, Amazon Alexa, Facebook e outros. Sua estrutura é baseada em fluxos de diálogos. Os bots podem trabalhar em assuntos diferentes, gerenciam os padrões de entradas de mensagens (*intents*), e geram respostas que podem levar por um caminho específico ou ramificado de perguntas e respostas, lembrando um fluxograma. O Dialogflow usa algoritmos da Google de inteligência artificial e aprendizado de máquina para generalizar os *intents*, aprendendo a encontrar padrões para responder de forma mais eficiente. Para alimentar os *intents*, basta adicionar um conjunto de exemplos de mensagens do usuário. O site do Dialogflow possui boa documentação em texto, imagem e vídeo sobre como trabalhar com a plataforma e boas práticas no processo da criação da base de conhecimento de um agente de conversação, *Conversation Design*. Há uma forma de proatividade por meio de eventos. Eles permitem que o bot invoque *intents* baseados em algo que acontece, ao invés de uma mensagem do usuário. Ele suporta eventos do Google Assistente e outras plataformas com base em ações realizadas pelo usuário [DIALOGFLOW 2019].

O **Pandorabots**[1] é um serviço online para construção e disponibilização de chatbots na web. O usuário adiciona elementos AIML na base de conversação e pode testar na própria plataforma, analisando os logs e identificando o que o bot não consegue responder para, então, adicionar novos elementos AIML. Em tempo, como os chatbots têm se tornado cada vez mais uma ferramenta de relevância no atendimento em grandes empresas, o mercado desse serviço tem atraído diversos investidores, *startups* e empresas de desenvolvimento. Sendo assim, há uma ampla gama de exemplos que ainda poderiam ser citados.

3 Plataforma Arisa Nest

A Plataforma Arisa Nest (Figura 1) é uma evolução da implementação apresentada em [Zambiasi e Rabelo 2012]. Ela é hoje uma ferramenta *PaaS*, onde os usuários podem hospedar assistentes virtuais e gerenciá-los por meio de uma interface web. Ela possui uma estrutura de criação de diálogos orientados a contextos para alimentar a base de conversação, um editor de scripts em linguagem de programação Lua que pode ser usado para complementar as respostas dos diálogos com resultados da execução de algoritmos, serviços web no padrão SOAP ou REST podem ser consumidos pelos scripts para executar coisas de fora da plataforma, gerenciamento de crenças com informações dos usuários, comportamentos em Lua que podem ser agendados e executados conforme mudança do estado do bot fornecendo proatividade. Ela permite a criação de vários bots e cada bot pode ter vários usuários colaboradores.

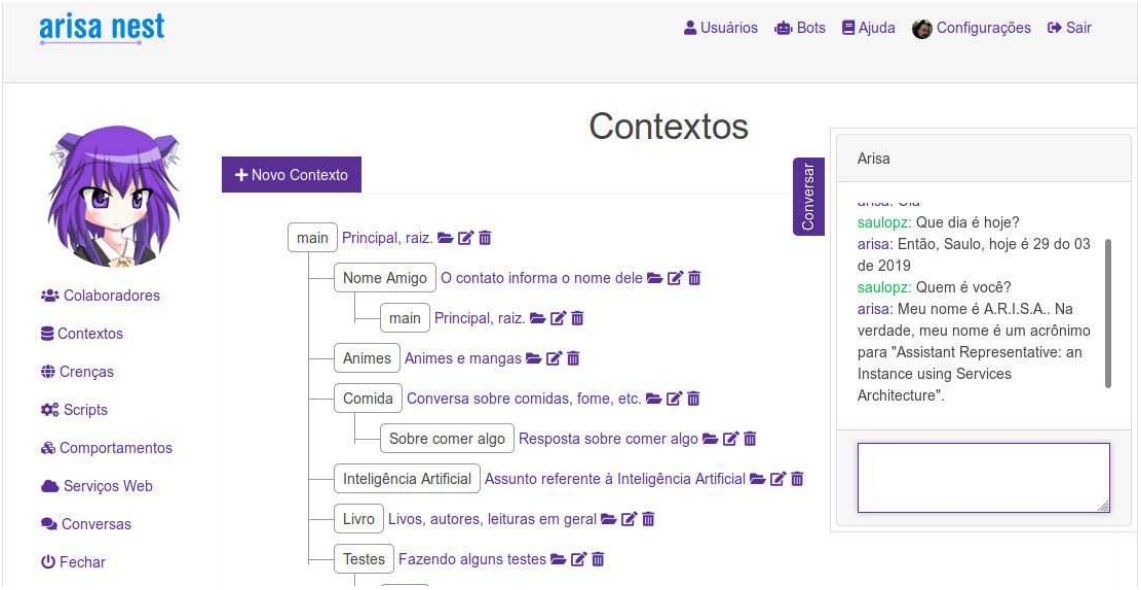

Figura 1. Interface da plataforma Arisa Nest, mostrando o menu superior, o menu do bot à esquerda, a árvore de contextos e a janela de chat para testes de conversas.

A base de conversação pode ser vista como uma árvore de contextos (Figura 1). Um contexto é usado para definir um assunto e pode-se mudar de assunto conforme a conversa com o bot. Por exemplo, a pergunta "o que você prefere?" é respondida de forma diferente para os contextos "comida" e "livro". É necessário haver um contexto inicial/pai chamado main, ou 0, e dentro dos contextos são criados os diálogos.

[1] https://home.pandorabots.com

Os diálogos servem para identificar uma mensagem do usuário e produzir uma resposta. O campo Condições efetua testes lógicos baseados nas suas crenças e nas crenças dos usuários para permitir ao bot usar ou não esse diálogo para responder uma mensagem. O campo Padrões é usado para identificar se o diálogo bate com a mensagem do usuário, no caso positivo, é escolhida uma resposta aleatória. É possível inserir vários padrões e respostas separados por ponto e vírgula. Um link e uma imagem também podem ser utilizados como parte da resposta. O campo "Ir Para" serve para direcionar o motor do chatbot para outro contexto. Há um tipo de diálogo especial, chamado Fuga, em que deixa-se o campo padrões vazio e, caso o bot não encontre nenhum diálogo que fecha naquele contexto, e existir um diálogo tipo fuga, ele escolhe uma resposta aleatória do mesmo. Nos padrões, o símbolo "%" representa um caractere curinga, que pode estar entre palavras, no final e/ou no início, conforme pode ser visto na Figura 2.

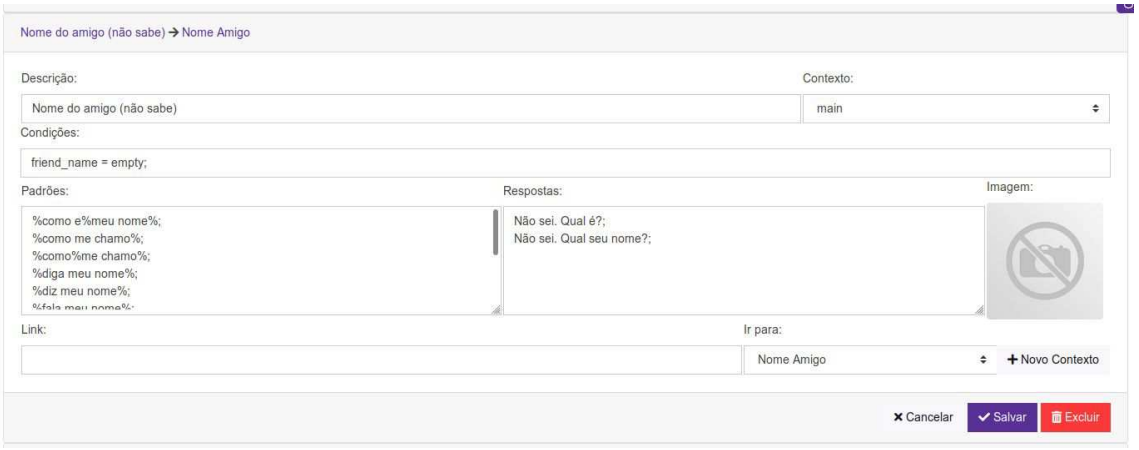

Figura 2. Formulário de diálogos

Os diálogos permitem alguns tipos especiais, em padrões e respostas, para gerir as crenças locais e globais e utilizar um algoritmo para complementar uma resposta. Por exemplo:

Padrão: %meu nome e **{friend_name};**
Resposta: Muito prazer **{friend_name}.** Me chamo **{bot_name};**
Mensagem usuário: Olá, meu nome é **Saulo**
Resposta do bot: Muito prazer **Saulo.** Me chamo **Arisa**

Nesse caso, o bot pega a informação no local de {friend_name} e armazena como uma crença local. {bot_name} é usando na resposta e provêm de uma crença global. Quando a informação não precisa ser armazenada, mas apenas usada para algo e depois descartada, coloca-se um sinal de $ antes do nome:

Padrão: %quanto e **{$1}** mais **{$2};**
Resposta: O resultado é **{@soma $1 $2};**
Mensagem usuário: Quanto é **2** mais **5**?
Resposta do bot: O resultado é **7**

As informações $1 e $2 são utilizadas apenas para o processamento nesse diálogo. Nele também que foi usado um sinal @. Ele indica a chamada ao script `soma` e

passa como parâmetro duas informações, 2 e 5. Os **scripts** são algoritmos desenvolvidos em linguagem de programação Lua na plataforma e podem ser usados de diversas formas, como executar cálculos e operações complexas, além de consumir serviços web externos à plataforma. Por exemplo, o script do diálogo anterior poderia ser apenas:

```
return tonumber(args[1]) + tonumber(args[2])
```

Ou o retorno de uma chamada a um serviço web:

```
return wscall('math', 'sum', { a = args[1], b = args[2] })
```

Em ambos os casos pode-se observar os argumentos de entrada `args`. No segundo caso, é usada a função `wscall`, que consome um serviço web cadastrado na plataforma chamado **math** usando a operação **sum**. No editor de scripts e comportamentos há um botão que, quando clicado, o usuário seleciona o serviço e a operação. O texto da chamada do serviço é adicionado automaticamente no editor, no cursor de edição, facilitando a criação do código.

Os **Comportamentos** são executados regularmente ou conforme agendamento configurável. Por exemplo, o comportamento da Figura 3 é agendado para todo dia 1º de Janeiro, às 0h01min desejando feliz ano novo para todos os seus contatos do telegram.

```
1  friends = json.decode(friendList())
2  for i = 1, #friends do
3      id = friends[i]
4      im = getLocal(id, 'im')
5      if im == 'telegram' then
6          message = getDialog('main', 'feliz_ano_novo', null)
7          sendMessage(id, message)
8      end
9  end
```

Figura 3. Comportamento feliz_ano_novo

A função `friendList` retorna os IDs de todos os contatos do usuário em uma string JSON. `getLocal` retorna uma crença de um contato, conforme seu **ID**. No caso, foi retornado o *instant messaging* (IM) usado pelo usuário. Caso seja 'telegram', utiliza uma resposta aleatória de um diálogo chamado **feliz_ano_novo** do contexto **main**, com a função `getDialog`, e envia como mensagem para o usuário com `sendMessage`. Algumas outras funções da plataforma que podem ser utilizadas tanto nos scripts quanto nos comportamentos são: `globalList` retorna a lista dos nomes das crenças globais; `localList` lista dos nomes das crenças locais de um contato; `getGlobal` retorna o valor de uma crença global; `setGlobal` altera uma crença global; `setLocal` altera uma crença local; `deleteGlobal` exclui uma crença global; `deleteLocal` exclui uma crença local; `last` retorna dia e horário da última mensagem enviada por um usuário ao bot; `sendToBot` envia uma mensagem para outro bot; `callRest` chama um serviço REST.

Há uma área de visualização das conversas, com informações das crenças locais e registros das conversas. Uma mensagem com cor diferente nas conversas indica que o bot usou uma resposta de fuga. Isso facilita para identificar mensagens que o bot não

está conseguindo responder. Do lado direito do plataforma, há um botão que permite abrir uma janela de chat para testar o bot na própria plataforma (Figura 1). A plataforma suporta multi linguagem e está disponível atualmente nos idiomas português e inglês.

3.1 Considerações sobre a Implementação

O motor do chatbot da Arisa Nest foi implementado na linguagem PHP e sua base de conhecimentos é armazenada em banco de dados Postgresql. O sistema de execução baseado em contextos inicia no contexto raiz *main*. A mensagem entrada do usuário é comparada com os padrões cadastrados nos diálogos daquele contexto. Caso algum padrão de um ou mais diálogos fechar com a entrada do usuário, é escolhido um diálogo aleatório, dentre os encontrados, e uma resposta aleatória desse. Antes do bot enviar uma resposta, a mensagem do usuário é quebrada em partes e as crenças são extraídas da mensagem e usadas para as crenças, para compor a resposta ou para serem utilizadas como parâmetros de algum script. Se o diálogo escolhido possui um link para outro contexto, o usuário é direcionado para ele. Quando o usuário envia uma nova mensagem, caso ela não encontre um diálogo no contexto atual, o motor navega para o contexto pai, e assim segue até que chegue ao contexto raiz. Se mesmo no contexto raiz, o motor não encontrar padrões que fecham com a mensagem, ele responde com uma resposta de "fuga". Os scripts são utilizados pelos diálogos, podem ler e escrever crenças, mandar mensagens para outros usuários ou bots, executar operações externas por meio de serviços web SOAP ou REST. Os comportamentos, por sua vez, são a forma dos bots agirem de forma proativa. Para que essa autonomia seja possível, é necessário que um programa executor de comportamentos. Neste caso, foi implementado um programa de execução de comportamentos em linguagem Go[2]. O programa executa em *background* no sistema operacional, verificando quais comportamentos estão configurados para execução e os executa, conforme configuração de agendamento. A interface do usuário foi implementada utilizando PHP, HTML, CSS, JavaScript, Ajax e Bootstrap 4. A plataforma encontra-se instalada e rodando em um servidor na nuvem com Sistema Operacional Ubuntu 18.04, servidor HTTP Apache 2 e banco de dados Postgresql, sob o domínio https://arisa.com.br/nest.

3.2 Arisa Nest como Plataforma para Agentes

A plataforma Arisa Nest permite a criação de múltiplos bots que podem executar algoritmos, consumir serviços web e podem conversar entre si para colaborar na resolução de problemas. Tal como os agentes [Russell e Norvig 2013] eles podem perceber os mais diversos possíveis ambientes, conforme funcionalidades e interoperabilidades fornecidas por meio do acesso à serviços web e mensagens de usuários ou outros bots (habilidade social). Armazenam crenças para compor seu estado interno, executam seu processamento interno por meio de scripts e algoritmos de comportamentos que podem alterar seu estado e agir de forma autônoma e proativa. Por meio desses recursos, é possível construir na plataforma agentes que vão desde os puramente reativos, até os cognitivos e/ou baseados em objetivos e utilidade.

[2] Linguagem de programação criada pela Google, baseada na linguagem Limbo do sistema operacional Inferno, focada em programação concorrente e produtividade. Em 2009 foi lançada como código livre e atualmente com licença BSD. https://golang.org/

Considera-se aqui o seguinte cenário fictício simplificado para uma prova de conceito. O funcionário Saulo é responsável pelo controle de estoques de produtos em uma empresa. Um agente foi criado para auxiliá-lo, verificando regularmente a situação em um sistema legado e, caso o estoque de um produto esteja com a quantidade menor do que uma certa quantidade pré configurada, o agente inicia um processo de compra, com a autorização do funcionário. Caso o processo de compra, seja autorizado, o agente entra em contato com um agente de vendas da empresa fornecedora e faz o pedido.

Algumas considerações sobre as atividades e comunicação entre os agentes na plataforma Arisa Nest: (i) a comunicação entre os agentes pode ser padronizada para facilitar o entendimento entre eles com palavras chaves e parâmetros ou pode-se utilizar mensagens em um formato padrão, como JSON; (ii) o reconhecimento de outro agente se dá pelo seu ID na plataforma e o servidor da plataforma, podendo o outro agente estar localizado em outro servidor que também possua a plataforma instalada; (iii) Um agente entra em contato com o outro por meio de mensagens e ela é analisada pelo motor de chatbot, que encontra o diálogo correspondente e inicia um script; (iv) Um processo automatizado ou conversação entre os agentes ou entre um agente e seu usuário pode ser iniciado por um comportamento ao verificar mudanças nas crenças ou por mudanças em uma informação consumida via um serviço web.

O algoritmo da Figura 4 é um comportamento agendado para executar a cada hora. Ele verifica o estoque e, caso necessário, cria uma ordem de compra, pedindo confirmação do usuário. Em caso afirmativo, efetua a compra.

```
1  produto.id = json.decode(wscall('estoque', 'baixo', null))
2  if produto.id ~= '' then
3      fornecedor = json.decode(wscall('fornecedor', 'get', { produto = produto.id }))
4      if fornecedor.id ~= '' then
5          ordemjson = wscall('ordem', 'novaCompra', { fornecedor = fornecedor.id,
6              produto = produto.id, quantidade = produto.quantidadeCompra })
7          setLocal(getGlobal('funcionario_id'), 'ordem', ordemjson)
8          ordem = json.decode(ordemjson)
9          mensagem = getDialog('estoque', 'envia_ordem_funcionario', { ordem.id, produto.nome,
10             produto.quantidadeCompra, fornecedor.nome, fornecedor.preco })
11         sendMessage(getGlobal('funcionario_id'), mensagem)
12     end
13 end
```

Figura 4. Comportamento verifica_estoque.

Seguindo sua execução, na linha 1 ele consome um serviço web de acesso a um sistema legado, que retorna informações de um produto se o mesmo está com o estoque baixo. Na linha 3 consome um serviço que encontra um fornecedor cadastrado que trabalha com aquele produto e, caso exista mais de um, faz uma escolha conforme critérios daquele serviço. Nas linhas 4 e 5 é criada uma ordem de compra em um sistema legado, e as informações retornam no formato JSON. Na linha 7, as informações dessa ordem são armazenadas como crenças do usuário, pois a mesma ainda necessita de confirmação e o usuário pode perguntar mais informações sobre a mesma antes de confirmar. Nas linhas 9 e 10 é criada uma mensagem para o usuário com base no diálogo de nome `envia_ordem_funcionario` do contexto `ordem`, passando os parâmetros necessários (Figura 5). Na linha 11, a mensagem é enviada para o funcionário responsável (Figura 6).

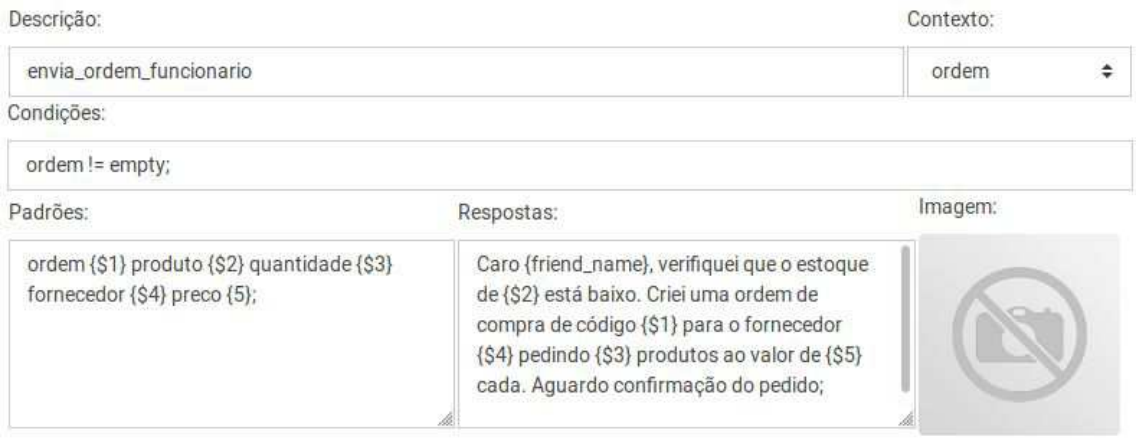

Figura 5. Diálogo envia_ordem_funcionario.

Quando o usuário envia a mensagem confirmando ou cancelando a ordem, é acionado um script que envia uma mensagem ao agente fornecedor. Caso o fornecedor não puder cumprir com a venda, a ordem é cancelada e o usuário é informado, caso contrário a compra é efetivada (Figura 6).

Mensagem Arisa para fornecedor:

```
pedido ordem 2923 produto 213 quantidade 10
```

Três possíveis respostas do fornecedor:

```
venda 2923 ok| venda 2923 em falta| venda 2923 produto inexistente
```

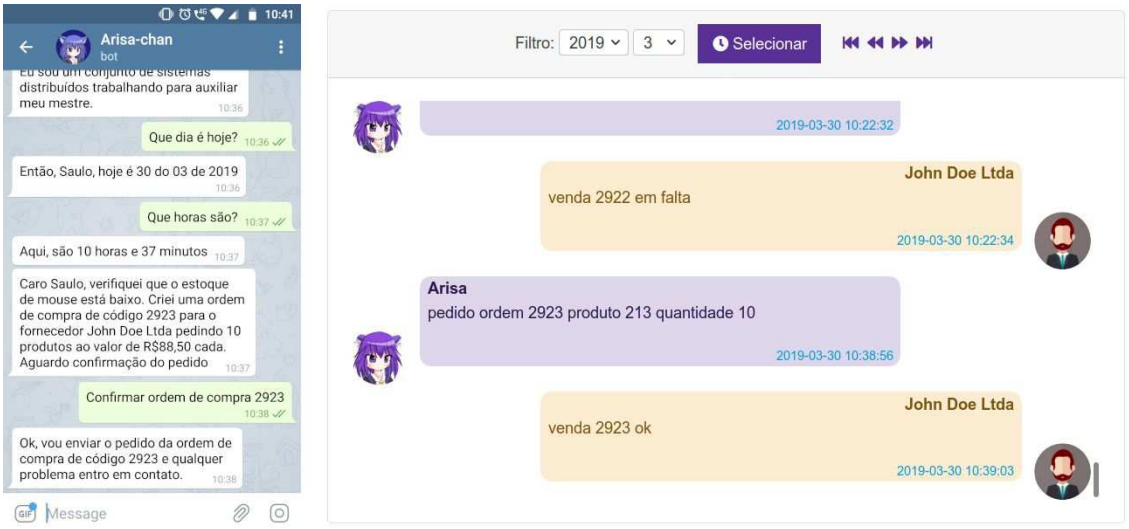

Figura 6. Esquerda: interação com o usuário via telegram; Direita: registro da interação entre os bots na plataforma.

Ao interagirem, os bots alteram seus estados internos por meio das crenças, scripts e comportamentos. Eles podem interagir tanto com seus usuários como com outros bots para alcançar seus objetivos. Mesmo que a plataforma possa ter vários bots executando ao mesmo tempo, cada bot precisa ser cadastrado e configurado individualmente. No momento a plataforma não permite a criação e exclusão dinâmica

de bots como para um sistema multiagentes dinâmico, mas tal recurso já encontra-se em desenvolvimento.

3.3 Considerações da Plataforma

Comparando a plataforma Arisa Nest com as ferramentas citadas na seção 2, a ELIZA, ALICE e Pandorabots são apenas agentes de conversação e não executam tarefas e pouco fazem no caso de adquirir informações dos usuários. No caso da ALICE e Pandorabots, há o uso de uma linguagem AIML, já bastante madura, mas trabalhar com essa linguagem não é tão intuitivo e requer um certo aprendizado e conhecimento, enquanto a alimentação dos diálogos da Arisa Nest é mais simples e requer apenas entender a estrutura dos padrões e crenças. Já no caso do Watson Conversation, ele tem uma interface de fácil aprendizado e de uso relativamente intuitivo, não requer entender padrões dos *intents*, além de utilizar algoritmos de inteligência artificial do Watson para identificar padrões de entrada parecidos com os *intents* criados pelo usuário para poder generalizar. Entretanto, ele não executa algoritmos criados pelo usuário, nem tem proatividade e seu foco é na conversação. O Dialogflow é uma ferramenta bem aprimorada para trabalhar com agentes de conversação e é orientado a fluxos de conversas estilo fluxograma, algo que poderia se comparar em parte com os contextos da Arisa Nest. Ele possui também um recurso de proatividade através de eventos, mas ele serve para iniciar um contato com o usuário por algum fluxo de diálogo. A forma dele trabalhar com execução de algum algoritmo é por meio do recurso de *fulfillment*, sendo necessário a configuração de um *webhook* para executar um script em javascript sob Node.js. Entretanto, mesmo se mantendo nos padrões da grande estrutura da Google, há uma certa complexidade para se trabalhar com os scripts e requer um conhecimento mais aprimorado. Para trabalhar com scripts na Arisa Nest basta conhecer as funções de acesso à plataforma, um pouco da linguagem de programação Lua, largamente utilizada em jogos digitais. Além disso, eles podem ser escritos, testados e executados na própria plataforma. A Alexa é uma plataforma bastante completa e tem a vantagem de utilizar toda a estrutura da Amazon, incluindo o AWS para a criação de scripts, hardwares e dispositivos. Ela já vem com uma grande gama de *skills* e o usuário pode adicionar outros, via ferramenta de compartilhamento ou produzidas pelo próprio usuário. Também há proatividade por meio de eventos. Contudo, seu nível de complexidade está no mesmo nível, ou maior, que o Dialogflow.

4 Casos e Testes

A Arisa Nest foi testada e avaliada em quatro casos. O primeiro foi um projeto de estágio no curso de Engenharia de Controle e Automação da UFSC, em 2018, pelo aluno Ricardo Ventura e aplicado na secretaria acadêmica do curso de Pós-Graduação em Engenharia Mecânica (PosMec) [Ventura 2018]. Para este projeto foi utilizada a versão anterior da plataforma, apenas com os recursos de conversação via árvore de contextos. O aluno passou por um breve treinamento para a utilização do sistema, criação de contextos, diálogos, estrutura dos padrões e criou uma base de conversação sem maiores problemas. A maior dificuldade foi na aquisição, filtragem e estruturação das informações para inserir no sistema. O bot foi avaliado pelas respostas preenchidas em um formulário na interface web do bot e, conforme os resultados, para 70% dos usuários, o bot respondeu corretamente aos questionamentos, 60% responderam que

tornou mais fácil o acesso à informação da PosMec e para 60% o sistema contribuiu para padronizar as informações. Atualmente o bot ainda encontra-se em funcionamento e foi migrado para a versão atual da plataforma Arisa Nest.

O segundo caso foi o bot Riunita, desenvolvido em 2017 e 2018 como projeto de pesquisa do Artigo 170[3] pelo acadêmico Laércio de Sant'Anna Filho para o Repositório Institucional (RIUNI[4]) da Universidade do Sul de Santa Catarina [Sant'Anna Filho 2018]. Foi criado um módulo no canto inferior do site do RIUNI. O projeto focou na resolução de dúvidas dos usuários. Houve um treinamento de 30 minutos para a utilização da plataforma e formas de criação dos contextos e diálogos. Mas, foi na prática que o aluno compreendeu a estrutura de uma base de conversação baseada em padrões de entrada e contextos. Atualmente o bot se encontra em funcionamento no Repositório Institucional e os resultados do projeto foram apresentados na Jornada Unisul de Iniciação Científica (JUNIC) de 2018 [UnisulHoje, 2018].

O terceiro caso foi o CMBOT, desenvolvido pelos acadêmicos Itamar Ghidini e Winicius Mattos [Ghidini e Mattos 2018] como Trabalho de Conclusão de Curso (TCC) no curso de Sistemas de Informação na Universidade do Sul de Santa Catarina (UNISUL) e aplicado no ambiente de desenvolvimento da empresa em que um dos alunos envolvidos no TCC era funcionário. Sua finalidade seria recepcionar o cliente (usuário dos sistemas da empresa) e auxiliá-lo a completar o atendimento até seu fechamento. No site de atendimento foi criado um módulo web no canto inferior direito do site para interagir com o bot. O projeto foi desenvolvido na versão atual da plataforma, mas, além da base de conversação, usaram apenas o recurso de crenças para tratar algumas informações dos clientes. Foi dado um pequeno treinamento aos alunos, mas eles tiveram acesso a um tutorial na web, facilitando o aprendizado. O maior problema encontrado por eles foi a falta de documentação interna e dos atendimentos feitos pelos funcionários, indicando a necessidade de se documentar o processo e as vantagens de haver um sistema que gere registros automatizados, como no caso do chatbot. O sistema foi avaliado por um conjunto de usuários (6 desenvolvedores e 3 atendentes de suporte) e foi aplicado um questionário. Conforme os resultados, 78% responderam que o chatbot deixa a forma de atendimento mais eficiente e assertiva, 66% concordaram que a ferramenta facilita, pois permite a abertura de chamado fora do horário do trabalho, 66% disseram que o bot deve ser proativo para interagir com o cliente no caso dele ficar muito tempo na seção sem atividade, 100% concordaram que a tecnologia é importante para o atendimento, resolvendo pelo menos as dúvidas mais simples do cliente. Por fim, 89% disseram que a base de conhecimento está totalmente relacionada a eficiência, o que demonstra a importância de uma boa construção dessa base. Foi também identificado que para os atendimentos mais diretos e simples foi preferido o uso do chatbot pela agilidade e disponibilidade da ferramenta, sem a necessidade da espera em fila do atendimento.

O quarto caso foi o Projeto de Final de Curso (PFC) do aluno Ricardo Ventura [Ventura 2019], do curso de Engenharia de Controle e Automação da UFSC e se deu por uma evolução do seu estágio acadêmico. A avaliação produzida naquele projeto

[3] Artigo 170: programa de bolsas de pesquisas da Secretaria da Educação do Estado de Santa Catarina para incentivar potenciais talentos entre alunos de cursos de graduação com um grau de carência econômica e financeira.

[4] RIUNI: repositório de produções intelectuais da comunidade acadêmica da Unisul

trouxe questionamentos para melhorias no processo de atendimento aos usuários de chatbots. O aluno usou todos recursos disponíveis na atual versão da plataforma Arisa Nest, indo mais no viés da assistência virtual do que especificamente de um chatbot. A intenção foi criar um bot para executar ações para os usuários, automatizar processos e, inclusive, com integração à serviços web padrão SOAP, desenvolvidos pelo próprio aluno, providos pela UFSC e por outras empresas ou ecossistemas.

Após analisar algumas situações, foram modelados os processos, definindo os diálogos, as crenças, scripts e implementação dos comportamentos. Entre algumas atividades do bot estão: informar o índice de aproveitamento semestral acumulado do aluno, as aulas de um dia na semana, cardápio do dia no restaurante universitário, a nota das disciplinas cursadas, alterar e-mail cadastrado, fazer matrícula, fornecer histórico semestral e, como proatividade, informar diariamente as aulas do usuário do dia seguinte. Outra atividade do bot, mais para um coordenador de estágios e PFC, foi automatizar o processo analisar alguns critérios do termo de compromisso de estágio em que é necessário pesquisar informações em diferentes locais, findando o resultado das análises com o envio de um e-mail ao coordenador e ao aluno analisado.

Também foi criado um outro cenário para automatizar parte do atendimento aos clientes no processo de venda em uma empresa fictícia. Este cenário está mais para testes na plataforma para verificar a aplicabilidade de cenários do Operador 4.0 na mesma. Este se daria pela utilização de assistente virtual para auxiliar um operador de equipamentos industriais em um ambiente inteligente, nos moldes da indústria 4.0, em que o assistente pode tomar algumas decisões e realizar algumas tarefas de forma independente. Esse cenário foi uma adaptação simplificada de um caso mais complexo abordado no artigo "*Softbots Supporting the Operator 4.0 at Smart Factory Environment*" [Rabelo, Romero e Zambiasi 2018]. Segundo Ventura [2019], com a utilização dos recursos da plataforma, a implementação foi simples. O bot recebia uma informação que iniciava o processo por meio da execução de um script. A busca de informações se dava através de chamadas na camada de serviços e, conforme a orquestração dos serviços e atividades implementadas no script Lua, o resultado cumpriu com as diretrizes pré-programadas e concluindo a tarefa. Para ele, "todas as funcionalidades da plataforma Arisa Nest se mostraram bastante robustas e complementares entre si. A possibilidade de consumir serviços web, executar scripts, armazenar crenças" e automatização com os comportamentos possibilita "uma grande variedade de implementações em diferentes cenários".

Atualmente a plataforma está sendo utilizada para quatro outros projetos em andamento: um PFC do curso de Automação da UFSC pelo acadêmico Murilo Rodrigues Padilha Leite, continuação do projeto do Ricardo Ventura na PosMec; um Projeto PUIC do aluno João Ricardo dos Santos Kleine Buckstegge na UNISUL; um projeto de mestrado no Programa de Pós-Graduação em Engenharia de Automação e Sistemas (PGEAS) na UFSC pelo aluno Brunno Abner Machado; e em um teste de utilização de assistentes virtuais para o PJe (Processo Judicial Eletrônico) do Conselho Nacional de Justiça (CNJ).

5 Conclusões e Trabalhos Futuros

Este artigo apresentou uma plataforma de assistentes virtuais chamada Arisa Nest, no modelo arquitetural *Platform as a Service* (PaaS) de disponibilização baseado na

nuvem, que provê recursos de gerenciamento de uma base de conversação, gerenciamento de informações na forma de crenças, scripts para execução de algoritmos, acesso a serviços externos padrão SOAP e REST e comportamentos proativos. Foram apresentados alguns exemplos de plataformas atuais no mercado para identificar alguns recursos desses tipos de ferramenta e para fazer uma análise comparativa com a plataforma Arisa Nest.

Foi feita uma apresentação da plataforma, alguns recursos, maneira de utilizá-los e, como proposta complementar, um caso foi apresentado como forma de utilização da plataforma também como um ambiente de criação de agentes, suportando todo seu ciclo de vida, comportamento e acesso a outros recursos, dispositivos e informações por meio do consumo de serviços web. Em seguida, foram apresentados alguns casos que utilizaram a plataforma como base para projetos de pesquisa e trabalhos de fim de curso de graduação. Cada caso foi avaliado em termos da utilização da plataforma. No contexto do que foi apresentado, testado e avaliado, a Plataforma se mostrou como uma ferramenta com certa maturidade e robustez, tanto para chatbots como para assistentes virtuais e agentes, resolvendo os casos testados de forma satisfatória.

A Arisa Nest é uma ferramenta que se encontra em constante evolução, serve como ambiente de pesquisa, tanto nas teorias e tecnologias utilizadas para seu desenvolvimento com a agregação de algoritmos e novos recursos, como para os projetos que a utilizam como plataforma para assistentes virtuais. Quanto à utilização da plataforma, alguns projetos já citados encontram-se em processo de criação e avaliação. Quanto ao seu desenvolvimento em si, ainda estão sendo feitas várias implementações para que a plataforma fique mais fácil de usar, criação de mais funções de acesso aos recursos da plataforma pelos scripts e comportamentos, funções para o bot criar, excluir e editar contextos, diálogos e crenças da sua própria base de conhecimento, se auto-clonar, criar outros bots e migrar para outros servidores e, ainda em análise, a adição de um recurso/editor de comportamentos baseado em agentes BDI (*Beliefs, Desire, Intentions*).

Referências

Abu Shawar, B.; Atwell E.. (2015) “ALICE Chatbot: Trials and Outputs”. Comp. y Sist., México, v. 19, n. 4, p. 625-632.

ALEXA. (2019) “Developer documentation: browse the technical documentation for Alexa, Amazon Appstore, and devices”, https://developer.amazon.com/documentation/, Março/2019.

AWS. (2019) “AWS Lambda”, https://aws.amazon.com/pt/lambda/, Março/2019.

DIALOGFLOW. (2019) “Dialogflow: build natural and rich conversational experiences”, https://dialogflow.com, Março/2019.

Ghidini, I.; Mattos, W.W.. (2018) “Desenvolvimento e aplicação de um chatbot para auxiliar o atendimento ao cliente”. Universidade do Sul de Santa Catarina, Sistemas de Informação, Trabalho de Conclusão de Curso. https://www.riuni.unisul.br/handle/12345/5986, Março/2019.

Markoff, J.. (2008) “A Software Secretary That Takes Charge”, In: New York Times, http://www.nytimes.com/2008/12/14/business/14stream.html?_r=1&scp=7&sq=personal%20assistant&st=cse, Março/2019.

Mazon, S.. (2018) “Desenvolvendo chatbots com Watson conversation”, https://www.ibm.com/developerworks/br/library/desenvolvendo-chatbots-com-watson-conversation/index.html, Março/2019.

Rabelo, R.J.; Romero, D; Zambiasi, S.P.. (2018) “Softbots Supporting the Operator 4.0 at Smart Factory Environments”. Moon I., Lee G., Park J., Kiritsis D., von Cieminski G. (eds) Advances in Production Management Systems. Smart Manufacturing for Industry 4.0. APMS 2018. IFIP Advances in Information and Communication Technology, vol 536. Springer, Cham.

Russel, S.; Norvig.. (2013) “Inteligência Artificial”. 3ª Ed. Tradução da terceira edição. Rio de Janeiro: Editora Campus / Elsevier.

Sant’Anna Filho, L. (2018) “Estratégia para apoiar o povoamento do repositório institucional de uma universidade de santa catarina utilizando um assistente virtual”, Universidade do Sul de Santa Catarina, Sistemas de Informação, Relatório de Estágio.

Tingiris, S.. (2018) “Amazon Alexa development 101”, https://youtu.be/QkbXjknPoXc, Março/2019.

UnisulHoje. (2018) “Assistente digital que auxilia repositório institucional é apresentada na Junic”, http://hoje.unisul.br/assistente-digital-que-auxilia-repositorio-institucional-e-apresentada-na-junic/, Março/2019.

Weizenbaum, J.. (1966) "Eliza - a computer program for the study of natural language communication between man and machine". Communications of the ACM, ACM, v. 9, n. 1, p. 36–45.

Ventura, R.. (2018) “Desenvolvimento de um chatbot para apoio a secretarias de cursos de universidades: um caso no programa de Pós-Graduação em Engenharia Mecânica”. Universidade Federal de Santa Catarina, Departamento de Automação e Sistemas. Relatório de Estágio.

Ventura, R.. (2019) “Desenvolvimento de um assistente virtual híbrido com propriedades de chatbot e de automação de processos de negócios”, Universidade Federal de Santa Catarina, Departamento de Automação e Sistemas, Projeto de Final de Curso.

Zambiasi, S.P.; Rabelo, R.J.. (2012) A Proposal for Reference Architecture for Personal Assistant Software based on SOA. IEEE Latin America Transactions, 10(1):1227-1234.