PUC

ISSN 0103-9741

Monografias em Ciência da Computação
No.07/2018

An Agent based Software Framework for Creating Domain Conversational Agents

Pedro Elkind Velmovitsky

Ruy Luiz Milidiú

Marx Viana

Carlos José Pereira de Lucena

Catarina Chagas

Stevens Rehen

Departamento de Informática

Monografias em Ciência da Computação, No. 07/2018 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena July, 2018

iBot: An Agent based Software Framework for Creating Domain Conversational Agents

Pedro Elkind Velmovitsky[1] Ruy Luiz Milidiú[1] Marx Viana[1]

Carlos José Pereira de Lucena[1] Catarina Chagas[2] Stevens Rehen[2,3]

[1]Laboratory of Software Engineering (LES), Pontifical Catholic University (PUC-Rio), Rio de Janeiro, Brazil

[2]Instituto D'Or de Pesquisa e Ensino (IDOR), Rio de Janeiro, Brazil

[3]Instituto de Ciências Biomédicas (ICB), Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil

{pvelmovitsky, mleles, lucena, milidiu}@inf.puc-rio.br

Abstract. Conversational Agents (chatbots) are computer programs that interact with users using natural language. Since its inception, the technology has advanced greatly and cloud based platforms from big companies allow developers to create intelligent and efficient chatbots. However, there are not many development approaches to the main modules of a chatbot that are flexible enough to allow the creation of different applications for each domain, while maintaining a robust dialogue control in the application. A promising approach for such a problem is the use of multiagent systems to distribute and perform the tasks performed by the chatbot. This paper introduces a general and flexible framework based on multiagent systems, which will facilitate building chatbots in any domain chosen by the developer, with dialogue control in the application. We conducted a research to outline relevant work and gaps for building a new architecture, which we based our software framework. We show how our approach allows creating applications based on the information state approach to dialogue management in order to increase flexibility and reuse. Our analysis suggests that the tool can be used to build different domains chatbots that allow a robust control of the information in the dialogue. In addition, the use of multiagent systems and the information state approach provided modularity and flexibility to the developed systems.

Keywords: Conversational Agents; Chatbots; Multiagent Systems; Dialogue Manager; Information State.

Resumo. Chatbots são programas de computador que interagem com usuários utilizando linguagem natural. Desde sua origem, a tecnologia avançou significantemente e aplicações baseadas na nuvem de grandes empresas permitiram que desenvolvedores criassem chatbots inteligentes e eficientes. No entanto, não há muitas abordagens de desenvolvimento aos principais módulos de um chatbot que são flexíveis o suficiente para permitir a criação de chatbots diferentes para cada domínio, mantendo um robusto controle de diálogo na aplicação. Existem trabalhos que tentam desenvolver uma abordagem mais flexível, cada um com suas vantagens e desvantagens. Uma das vantagens mais notáveis é o uso de sistemas multiagentes para distribuir e realizar tarefas feitas por chatbots. Nesse contexto, este trabalho propõe um framework geral e flexível baseado em sistemas multiagentes para construir chatbots em um domínio escolhido pelo desenvolvedor, com controle de diálogo na aplicação. Esta solução usa uma adaptação da abordagem de estado da informação, e agentes de software, para gestão do diálogo.

Palavras-chave: Chatbots; Sistemas Multiagentes; Gestão de Diálogo; Estado da Informação; Agentes Conversacionais.

In charge of publications

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br

Table of Contents

1 Introduction

A conversational agent, or chatbot, can be defined as a computer program that interacts with users using natural language (WOUDENBERG, 2014) (GATTI de BAYSER & CAVALIN, 2017). Nowadays, the technology has become very popular, due to the success of messaging apps and advances in Artificial Intelligence (APPEL, 2018). The concept of conversational agent can be first attributed to Alan Turing, who wrote his seminal work about machine intelligence and thinking in 1950 (TURING, 1950). In fact, it was a chatbot program, called ELIZA, that first passed a version of the Turing Test in 1966 by simulating the behavior of a Rogerian psychologist. With advances in Natural Language Understanding (NLU) and Machine Learning (ML) techniques chatbot technology has evolved since those days, as evidenced by a surge in the development of increasingly intelligent software in research and in business.

However, even though there have been great advancements in NLU, as evidences by cloud based solutions such as DialogFlow ("Dialogflow – Basics", 2018) and IBM Watson (WHITE, 2018), applications are still far from perfect, as most of them generate good results in specific domains; a general model for interpreting each and every utterance, independent of context, is still unattainable. Therefore, to deliver the best experience for the user, developers must consider building their chatbots using an efficient architecture that ensures the best solution for their domains. With this, considering the complexity and interaction of the different components involved in building chatbots, a promising approach is by using multiagent systems (MASs) in their development, especially in performing different tasks.

While there is a lot of research integrating dialogue systems and chatbot architecture with multiagent systems, most of these architectures usually deal with one specific domain, or few correlated domains. Such works try to increase the domains handled by distributing tasks to specific agents. However, an emerging challenge comes considering these works are not flexible enough to allow for the development of chatbots in different domains, or do not provide a robust dialogue control throughout the application.

In response to this challenge, we present an agent based Software Framework for Creating Domain Conversational Agents (iBot). It is a general and flexible framework based on multiagent systems, which will facilitate building chatbots in any domain chosen by the developer, with dialogue control in the application. This approach aims at providing new resources for the developer creating domain conversational agents. As such, more human characteristics can be considered in order to improve the deliberation process. By using these framework, it is possible to build chatbots that: (i) distribute specific tasks to software agents, increasing the system's intelligence and (ii) use the information state approach to dialogue management, thus allowing modularity, reuse and dialogue control.

The remainder of this paper is organized as follows: Section II provides background information about multiagent systems. Section III presents information about conversational agents' history and main components and Section IV describes the related work. Section V presents the proposed framework solution and Section VI presents the implementation of a proof of concept. Finally, Section VII presents conclusions and future work.

2 Multi-Agent Systems

2.1 Software Agents

Software agents are reactive systems that exhibit some degree of autonomy in achieving a goal: an agent is capable of independent action in unpredictable and changing environments, without the need of direct human intervention architecture (BORDINI, HUBNER and WOOLRIDGE, 2007). The system is called an "agent" because it is action oriented: an agent should actively pursue its goals and tasks, independently reasoning about the best way to do so. An agent is not usually found alone in a system; in fact, individual agents interact with each other, collaborating to perform complex tasks and achieve their respective objectives (BORDINI, HUBNER and WOOLRIDGE, 2007).

Among the main characteristics of a software agent, we can cite (BORDINI, HUBNER and WOOLRIDGE, 2007):

Autonomy: agents must act without direct human intervention, according to its reasoning;

Reactivity: agents perceive their environment and respond to changes in it;

Pro-activeness: agents should be opportunist and goal oriented, looking to execute actions when applicable to their goals;

Social: agents are capable of interacting with other agents, when appropriate, in order to achieve their goals.

2.2 The BDI Model

There are many ways to model the reasoning and behavior of agents. The most popular and researched approach is the BDI (Belief Desire Intention) model. To talk about this model, according to (BORDINI, HUBNER and WOOLRIDGE, 2007), we need to address the idea that we can talk about computer programs as if agents had a "mental state". Thus, when we talk about a Belief Desire Intention system, we are talking about computer programs with computational analogues of beliefs, desires and intentions.

Beliefs represent the agent's information about the environment. This information, however, is something the agent believes in but it may not be necessarily true. **Desires** represent the possible states of affairs that the agent might like to accomplish. That does not mean, however, the agent will act upon it – it is a potential influencer of the agent's actions. **Intentions** represent the state of affairs that the agent has decided to act upon. In other words, intentions can be considered as a selected option between the potential set of options/desires that the agent has decided to pursue.

These characteristics are the key data structures of the BDI model. The decision making approach used by the agent, therefore, is practical reasoning: the agent weighs conflicting information for and against the available options, according to its beliefs and desires. The result of this deliberation is the adoption of intentions, which in turn will lead to the execution of actions.

3 Conversational Agents

3.1 History and Early Chatbots

In 1950, Alan Turing published his seminal work about machine intelligence and thinking (TURING, 1950). To answer the question of whether machines can think, Turing proposed a test which he called the "imitation game" — nowadays known as the Turing Test — in which an interrogator asked questions to a human and to a machine, aiming to identify which of the two is the machine. If the interrogator is unable to do so, it is established that the machine can think. By proposing a machine that can dialogue with humans in natural language, Turing was creating the very concept of a chatbot.

In 1966, researchers at the MIT created ELIZA, the first chatbot to pass a version of the Turing Test. ELIZA (WEIZENBAUM, 1976) (WOUDENBERG, 2014) simulated a Rogerian psychologist whose goal was to make people reflect about their current situation by using techniques to keep the patient talking. Patients could not tell that they were talking to a program, to the surprise of ELIZA's creator, Joseph Weizenbaum.

In 1972, and inspired by ELIZA, the psychiatrist Kenneth Colby created PARRY (SHIEBER, 1994) (WOUDENBERG, 2014). Although this chatbot works similarly to ELIZA, using pattern matching techniques to produce a suitable output, PARRY had the goal of simulating a paranoid schizophrenic. Implementing a crude model of behavior and a conversational strategy, it was more advanced than its predecessor. Colby described it as "ELIZA with an attitude". PARRY also passed a variation of the Turing Test, in which psychiatrists were given transcripts of dialogues with PARRY and transcripts with actual paranoid schizophrenic patients, and were asked to decide which one was simulated. The psychiatrists did no better than random guessing in this test (SHIEBER, 1994) (WOUDENBERG, 2014).

Another noteworthy chatbot is ALICE, a chatbot created in 1995 by Dr. Richard Wallace. It has won the Loebner Contest — an annual competition to identify the most "human" computer and to award $100.000,00 for the first program that passes an unrestricted Turing Test — in 2000, 2001 and 2004 (WOUDENBERG, 2014) .

All these chatbots work similarly, using simple pattern matching and substitution techniques to process the input and produce an appropriate output. Some examples are shown in Table I (WOUDENBERG, 2014).

INPUT	**CHATBOT**
** you are (depressed | sad)*	*I AM SORRY TO HEAR YOU ARE \1*
** all **	*IN WHAT WAY*
** always **	*CAN YOU THINK OF A SPECIFIC EXAMPLE*

Table 1. Pattern Matching/Substitution Examples

3.2 Current Chatbot Technologies

Nowadays, chatbot technology has become very popular due to its integration in smartphones and smart devices (WEINBERGER, 2017). Examples of modern chatbots are Siri, embedded in Apple's iOS-based devices; Google Now, in Android devices; Amazon's Echo and Alexa; and Microsoft Cortana (GATTI de BAYSER & CAVALIN, 2017). These bots take advantage of advanced natural language understanding and machine learning techniques to generate responses based on analysis of web search results. Other modern chatbots use Statistical Machine Translation techniques to "translate" input into output responses.

Big players in the technology market, such as Google, Microsoft and IBM, have launched cloud based platforms, such as Google Cloud, Microsoft Azure and IBM Cloud, respectively, allowing access for developers to their services and solutions. Several of these deal with machine learning training problems, including natural language processing such as DialogFlow, Google's platform for processing natural language utterances and developing intelligent conversational agents.

Some applications, such as Facebook, Slack, Skype, Telegram, among others, allow chatbots to be hosted and deployed. Facebook Messenger, for example, had 34.000 developers on its platform and was hosting 30.000 bots in the end of 2016 (CAHN, 2017).

This allows chatbots developers to have great efficacy and efficiency in developing and deploying intelligent chatbot applications for specific domains and releasing them in different platforms.

3.3 Main Components of Conversational Agents

Building a chatbot requires several components, some of them specific to the domain and tasks being handled. However, chatbots systems usually have common modules. Even though there are different approaches to how they should be developed and implemented, the core concepts remain the same. These are listed below.

- **Natural Language Understanding**: it is responsible to receive the text and to output a semantic representation of its content, that can be read and understood by the Dialogue Manager component. If the utterance is spoken, a Speech Recognition module will identify the spoken words and convert them into text to the NLU.
- **Dialogue Manager**: it is responsible to control the state and flow of the conversation, storing discourse context and managing the different components in the architecture.
- **Task Manager**: it is responsible to actually perform the necessary tasks requested by the user to the bot.
- **Natural Language Generation**: it is responsible to receive a meaning representation of what to say from the Dialogue Manager and conveys this in natural language to the user. If the dialogue system is implemented with spoken dialogue as output, a Text-To-Speech component is necessary to take the generated text and transform it into synthetic speech.

3.4 Dialogue Management and the Information State Approach

The Dialogue Manager is the central component of a chatbot, with many different methods being developed to implement it, such as finite state, frame based, plan based or agent based systems. Each of these approaches are better suited for specific types of dialogue. For example, finite-state based systems are ideal for simple and non flexible dialogues where the system has the initiative, such as filling out forms, while plan based systems are more suited to mixed initiative dialogue.

The information state approach (BUCKLEY and BENZMÜLLER, 2005) proposes a unifying view of dialogue management, in which independent dialogue theories can be implemented and evaluated in a reusable foundation. Similar names for information state are "conversational score", "discourse context" or "mental state".

The term information state is used to define information about the conversation that is stored by the system. More specifically, (LARSSON & TRAUM, 2003) defines the information state of a dialogue as the information necessary to distinguish it from other dialogues, such as questions and answers, beliefs of the user, beliefs of the system, the last utterance and who performed it, among others.

As the dialogue progresses, the information state must be updated to reflect the consequences/effects that actions of the participants have on the dialogue context. In order to perform these updates, update rules are implemented to be fired in reaction to observed dialogue moves. These rules are specified by precondition rules and effects: preconditions define which information state is active at a time, and effect rules indicate the changes that must occur to achieve the new information state. Update rules may also have side conditions, allowing external functions to be called within the rule to calculate the transition.

The authors in (BUCKLEY and BENZMÜLLER, 2005) and (BUCKLEY and BENZMÜLLER, 2006) define a platform to implement information state based dialogue management using software agents, called ADMP, Agent based Dialogue Management Platform. Figure 1 provides the architecture of this platform. Information State is the central data structure of the system. It is made up of slots, which store values. These slots are read by software agents, called Update Rule Agents (URA). Each of these agents are associated with an update rule, which have in its preconditions a subset of the set of slots in the information state. When the URA observes that its preconditions hold, it computes the information state update encoded in the rule and writes the result to the update blackboard.

Update Agent (UA) surveys the update blackboard. After a timeout, or some other stimulus, it chooses the heuristically preferred IS Update, executes it, and resets the system for a new turn. Figure 2 shows the execution flow of an URA.

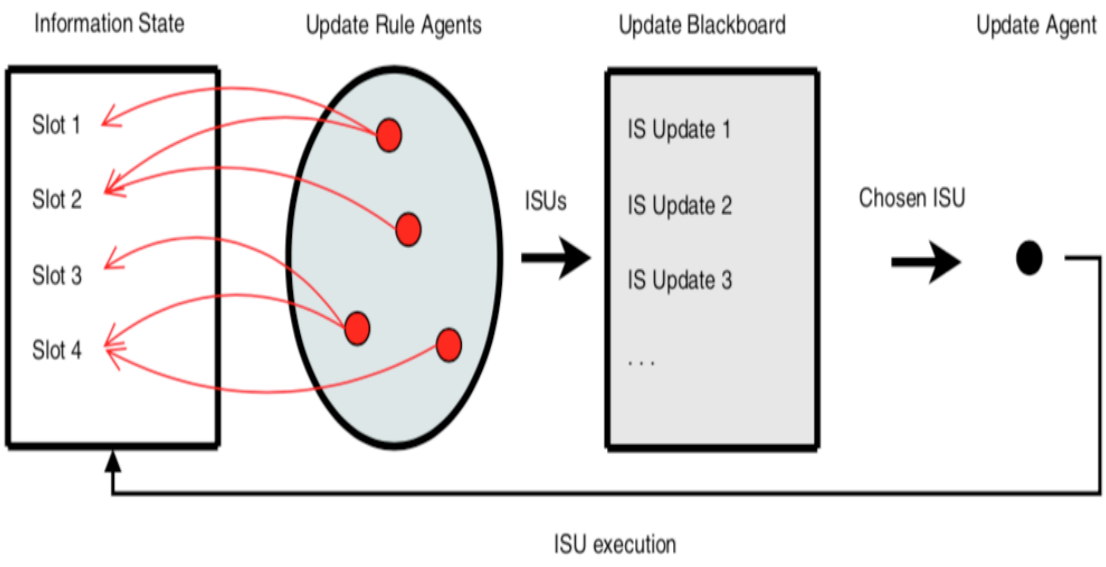

Figure 1. Architecture of ADMP

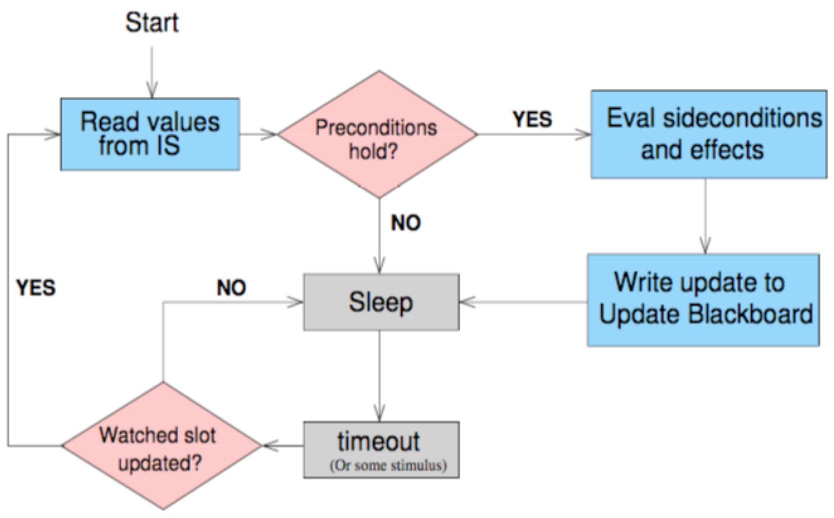

Figure 2. Execution flow of URA

4 Related Work

(CAHN, 2017) provides a literature overview of the basic concepts involved in building dialogue systems, while providing a case study of IBM Watson chatbot functionalities.

(WOUDENBERG, 2014) also reviews chatbot literature and develops a statistic tutor that uses the information state approach to dialogue management, coupled with pattern matching and substitution techniques. The authors in (BUCKLEY and BENZMÜLLER, 2005) and (BUCKLEY and BENZMÜLLER, 2006) define and expand upon ADMP, the agent based information state approach to dialogue management (see Section II D). These authors implement this platform in DIALOG, a system whose goal is to provide natural tutorial dialogue between a student and a mathematical assistance system.

The authors in (HO, NGUYEN & WOBCKE, 2006) implement a distributed architecture with multiagent systems to create a Smart Personal Assistant (SPA) to help users with e-mail and calendar tasks. In this model, a special Coordinator BDI agent mediates communication between the user and agents responsible for handling e-mail and calendar tasks. This agent is plan based and is also responsible for dialogue management in the application — it has plans, for example, to identify the conversational act and the user's intent in the conversation. This architecture is very interesting since it provides dialogue control through centralization in the Coordinator and distributes tasks to agents. However, the plan based agent is complex, and adding new domains, agents or new dialogue theories may be too complex. The information state approach could, in this case, provide for a solution that is more modular and flexible. In contrast to the centralization present in (HO, NGUYEN & WOBCKE, 2006), the authors in (LEE, LIN and WANG, 1999) propose a decentralized architecture in which each of the task agents is responsible for a part of the dialogue. A Facilitator module switches between agents according to the domain being spoken by the user, and each agent will then proceed with the dialogue, maintaining a record of its context. While this work avoids the complexities of implementing a centralized Coordinator, by distributing the dialogue it does not account for a robust dialogue control in the application.

5 Proposed Solution

Some approaches, such as (LEE, LIN and WANG, 1999) and (HO, NGUYEN & WOBCKE, 2006), show that distributing complex tasks in an agent society, which can autonomously realize specific tasks, can increase modularity and the overall power of the system by allowing more domains to be handled.

Moreover, some control in the dialogue is desirable, guaranteeing quality in the information and robustness of behavior. Therefore, we propose a software framework for building chatbots. This framework allows the creation of intelligent conversational agents that are able to perform complex tasks and maintain dialogue control while being flexible enough to allow the development of software in different domains.

Figure 3 presents the iBot framework. The boxes with dotted lines represent the hotspots; these are the components extended to implement the proof of concept, described in Section V. The other boxes contain the frozen spots. On the left side, the user interacts with the application through a *GUI* or a *User Interface Agent*. The architecture also accounts for *Natural Language Interpreter* and *Generator* components, to process the user's input according to the goals of the system and generate output. The *Dialogue Management* component is based on the information state approach implemented in ADMP. An optional *Persona* module has been added to this component. This is because, in some chatbots, developers may want to establish a persona in their program. For instance, (WALLIS, 2005) implements "Eugene the Cuttlefish", an embodied conversational agent that is vain and likes to be paid compliments about his colors; he may blush or get angry, and even withdraw for the conversation. In order to implement personalities like Eugene's, the *Persona* component can make alterations to the beliefs of the update agent in ADMP, altering its behavior. *The Task Agent Society* represents a multiagent system with software agents capable of performing the tasks in the domain. To implement a tour information service, for example, agents responsible for obtaining information about hotels, buses or the weather would be part of this society. They would also be able to access internal or external resources in order to accomplish their goals.

In a dialogue using this architecture, the user communicates with the *GUI*. The user input is passed to the *Dialogue Management* component, updating the information state. According to update rules, this component calls the *Natural Language Interpretater* module, which processes the input and extracts the relevant information from it, updating the relevant slots in the information state. These updates trigger the firing of new update rules, which may lead to other modules or components being called in turn. For example, if a slot in the information state stores the user's intention, and this is recognized as being "Obtain Bus Information", the update rule related to this slot will fire, and the necessary tasks accomplished. Those tasks may be to call the agent in the *Task Agent Society* responsible for obtaining information about buses. Once this information is returned, the update rule agent will place it on the blackboard, and the update agent will select it as the next dialogue move to be performed. The *Natural Language Generator*, then, generates a natural language output to provide the information that the user requested.

ADMP was chosen for dialogue management because it is based on the information state approach, thus allowing the implementation and evaluation of different theories of dialogue. Also, it does not constraint types and values in the slots of the information state, leaving the decision of what to model in the dialogue, i.e., which information will be important in the context of the domain, to the developer. This is particularly important considering that the proposed architecture needs to be flexible enough to allow for the development of applications in different domains. Each of these domains, in turn, may need different information to be modelled in the information state, and ADMP provides this flexibility. Finally, ADMP uses multiagent technology to implement the information state approach. Since the architecture uses software agents to increase modularity, in order to better perform complex tasks, the use of a multi-agent system makes for a modular and relatively simple Dialogue Manager — in contrast to the plan-based Coordinator in (LEE, LIN and WANG, 1999), for example — while also providing an interesting complement and a nice fit, such as adapting a persona to the update agent.

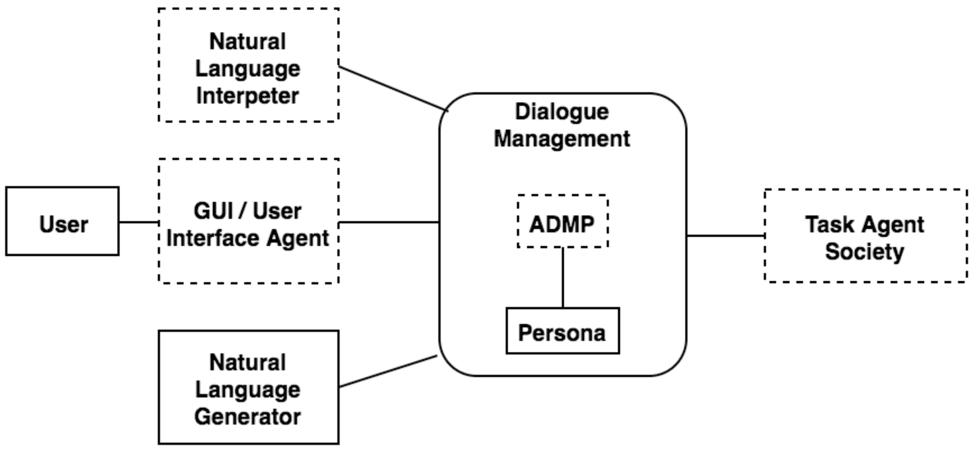
Natural Language Interpeter
Dialogue Management
User
GUI / User Interface Agent
ADMP
Task Agent Society
Persona
Natural Language Generator

Figure 3. iBot Framework

6 Proof of Concept

This work has been developed jointly with the Laboratory of Software Engineering, at PUC-Rio (LES) ("Les-Home", 2018), and IDOR ("IDOR", 2018), a non profit organization whose goal is to promote technological advancement in the healthcare area. One of these partnerships proposes the creation of a mobile app to stimulate scientific divulgation. Most specifically, the app has three main stakeholders: listeners, researchers and institutions/companies. Researchers and companies use the app to find each other and organize scientific events — for example, a university may wish to create an event about astronomy, while astronomers may want to share their knowledge with the public; so, they use the app to connect and create the event. Listeners, on the other hand, attend the events that they are interested in. The idea of the app came from the success of worldwide events such as Pint of Science, and research showed that the Brazilian people is interested in science — 61% of the interviewed declared being interested by the subject, a percentage larger than the one from the European Union — while they lack knowledge on the subject — 87% did not know the name of a Brazilian scientific institution, while 94% does not know the name of a Brazilian scientist ("Unicamp", 2017).

Chatbots can greatly improve the user experience of an app, including innovative ways to interact with users and solve their needs (GATTI de BAYSER & CAVALIN, 2017). Therefore, conversational agents for the proposed app have been developed as proof of concept for the framework. One of these is a login chatbot. It allows users to log in the app, storing their name and email. If the user is a company, then it should ask for the Employer Identification Number. If the user is a person, then it should ask for more information, such as Social Security and link to the person's CV, if the person is a researcher. Another proof of concept chatbot is a clarification chatbot for the subject of the event: if the user wants to know more about the subject of an event, he can ask the chatbot to explain it. The chatbot can give a more detailed and longer explanation if the user is experienced in the subject, or a simpler explanation, with analogies, to beginner users (the user profile can be inferred through the use of the app). The last proof of concept chatbot developed is an event chatbot, which allows users to research events in specific dates or about a specific subject and confirm presence in them. Table II shows an excerpt from the event chatbot dialogue.

Considering that these chatbots are to be integrated in a mobile app, the chatbots were developed for iOS, using Apple's IDE XCode ("Xcode - Apple Developer**"**, 2018**)**, and the programming language Swift 3.3. The GUI is a screen in an Apple mobile device, such as an iPhone. The chatbots accepts written and speech input and output, using native iOS development classes to generate speech. To interpret the user's natural language utterances, Google Cloud's DialogFlow platform was used ("DialogFlow – Basics",2018). DialogFlow allows the training of Machine Learning models according to samples of the user's utterances. It also provides an iOS SDK, where the developer makes a request to DialogFlow's API, which returns the user's intentions and parameters embedded in the utterance.

We extend ADMP in the dialogue management component by defining the Information State for the login chatbot with the following fields: (i) non-domain related: *Utterance_List, DFResponse, Intent_Lis, Next_Utterancet* (ii) domain-related: *name, email, id, interests, cvLink, user_type, person_type*. For the clarification chatbot, the non-domain related fields are the same, and the only domain-related field is *subject*, storing the subject to be explained, while the event chatbot – which also has the same non-domain related fields - has the domain-related field *event_list* to store the events according to the user's query.

The *Utterance_List* field stores the list of utterances spoken in the dialogue, and who spoke it. The *DFResponse* field stores the current response from DialogFlow, *Intent_List* stores the list of intents expressed in the dialogue by the user and *Next_Utterance* stores the next utterance by the system in the dialogue. The rest of the fields are domain related: *name* stores the username, *email* stores the user's email, *id* stores the user's Social Security or Employer Identification Number (depending on whether the user is a person or a company), *interests* store the user's interests, *cvLink* stores the link to the user's CV, *user_type* stores if the user is a person or a company and lastly, *person_type* stores if the user is a researcher, listener, or both.

SPEAKER	**DIALOGUE**
Chatbot	*Hello. I am a bot, here to help you to find events and confirm presence in events. Please, state the parameters of the event.*
User	*Find events about astronomy.*
Chatbot	*Start Date: 2018-07-12 End Date: 2018-07-20* *Start Date: : 2018-09-10 End Date: 2018-10-03* *Please state the event you would like to attend*
User	*The first.*
Chatbot	*You confirmed presence on the event starting 2018-07-12. Thank you!*

Table 2. Event Chatbot Dialogue Excerpt

The URA, defined in ADMP to monitor slots and trigger update rules, are created according to what intent they monitor in the *Intent_List* slot. For example, in the login chatbot a *NameAgent* monitors if the intent is *provideName*, and an *EmailAgent* monitors if the intent is *provideEmail*. Since the login chatbot does not need to perform any tasks outside collecting and storing user data, it does not use any other agent than the ones present in the ADMP architecture. For the clarification chatbot, the only URA is the *SubjectAgent* which monitors if the intent is *explainSubject*. This chatbot also has a *TaskAgent* which gets the subject's explanation in a database. The event chatbot has two URAs, *FindEventAgent* to find all the events which satisfy the user's query, and *SelectEventAgent* to confirm presence in an event. It also has a *TaskAgent* to get the events from a database. The agents where implemented using the *iMobile* framework for developing multiagent systems in iOS (SOUZA, MIRANDA and LUCENA, 2017). The complete dialogues of the login chatbot[1], the clarification chatbot[2] and the event chatbot[3] can be seen in the respective YouTube links.

7 Conclusion and Future Work

This paper presented the iBot framework, a framework for developing chatbots in different domains while maintaining a robust control of the information in the dialogue. This framework does not make any assumptions about which platform or programming language the developer must use, but rather provides a blueprint for developers to create chatbots in different domains, while maintaining a level of dialogue control in the application. Login, clarification and event chatbots have been implemented with the framework, to be integrated in the mobile app being developed jointly between LES and IDOR.

Future work will revolve around the improvement of the developed chatbots. Following the Chatbot Best Practices from IBM (CUMMINS, 2018), the conversational agents will be tested with users, since "supporting natural user interactions is the defining characteristic of the system". Also, they will be continuously monitored and tuned according to user feedback. Moreover, in order to further validate the solution, a qualitative evaluation of the framework will be performed. In this evaluation, users will be presented with theoretical use cases, where they will answer which agents and information states they would implement and reuse.

[1] https://youtu.be/BFpEnit6VOU
[2] https://youtu.be/m46oCJTqHLs
[3] https://youtu.be/VLIlo3FneIE

References

1. APPEL, Ana Paula et al. **Combining Textual Content and Structure to Improve Dialog Similarity**. arXiv preprint arXiv:1802.07117, 2018.
2. BORDINI, Rafael H.; HÜBNER, Jomi Fred; WOOLDRIDGE, Michael. **Programming multi-agent systems in AgentSpeak using Jason.** John Wiley & Sons, 2007.
3. BUCKLEY, Mark; BENZMÜLLER, Christoph. **An Agent-based Platform for Dialogue Management**. Proceedings of the Tenth ESSLLI Student Session, Edinburgh, Scotland, p. 33-45, 2005.
4. BUCKLEY, Mark; BENZMÜLLER, Christoph. **An agent-based architecture for dialogue systems**. In: International Andrei Ershov Memorial Conference on Perspectives of System Informatics. Springer, Berlin, Heidelberg, 2006. p. 135-147.
5. CAHN, Jack. **CHATBOT: Architecture, design, and development**. University of Pennsylvania School of Engineering and Applied Science Department of Computer and Information Science, 2017.
6. CAVALIN, P.; GATTI DE BAYSER, M. (2017) **Tutorial on Architectures and Algorithms for Conversational Agents**.16th International Conference on Autonomous Agents and Multi-agent Systems (AAMAS 2017).
7. CUMMINS, H. (2018). **Chatbot Best Practices - IBM Cloud Blog**. Retrieved April 30, 2018, from https://www.ibm.com/blogs/bluemix/2018/01/chatbot-best- practices
8. **Dialogflow - Basics.** (2018). Retrieved April 27, 2018, from: https://dialogflow.com/docs/getting-started/basics
9. **Instituto D'Or de Pesquisa e Ensino | IDOR.** (2018). Retrieved April 11, 2018, from: http://www.idor.org
10. **Les - Home**. (2018). Retrieved April 11, 2018, from: http://www.les.inf.puc-rio.br/ LEVINSON, S. (2016). Chapter 9: Speech Acts. In HUANG, Y. Oxford Handbook of Pragmatics.
11. LIN, Bor-shen; WANG, Hsin-min; LEE, Lin-shan. **A distributed architecture for cooperative spoken dialogue agents with coherent dialogue state and history.** 1999.

12. **Pesquisa revela que brasileiro gosta de ciência, mas sabe pouco sobre ela | Unicamp.** (2018). Retrieved April 11, 2018, from: https://www.unicamp.br/unicamp/ju/noticias/2017/09/25/pesquisa-revela-que- brasileiro-gosta-de-ciencia-mas-sabe-pouco-sobre-ela
13. SHIEBER, Stuart M. **Lessons from a restricted Turing test**. arXiv preprint cmp- lg/9404002, 1994.
14. E SOUZA, Pedro Augusto da Silva; MIRANDA, Andrew Diniz da Costa; DE LUCENA, Carlos José Pereira. **iMobile: A Framework to Implement Software Agents for the iOS Platform.** ICSEA 2017, p. 125, 2017.
15. TRAUM, David R.; LARSSON, Staffan. **The information state approach to dialogue management.** In: Current and new directions in discourse and dialogue. Springer, Dordrecht, 2003. p. 325-353.
16. TURING, Alan M. **Computing machinery and intelligence**. In: Parsing the Turing Test. Springer, Dordrecht, 2009. p. 23-65.
17. WALLIS, Peter. **Believable conversational agents: Introducing the intention map.** In: International conference on autonomous agents and multiagent systems. 2005. p. 17-22.
18. WEINBERGER, M. **Why Amazon's Echo is totally dominating — and what Google, Microsoft, and Apple have to do to catch up.** (2017). Retrieved February 28, 2018, from https://finance.yahoo.com/news/why-amazons-echo-totally- dominating-133000032.html
19. WEIZENBAUM, Joseph. **Computer power and human reason: From judgment to calculation**. 1976.
20. WHITE, C. (2018**). The Chatbot Never Sleeps: How We Created a Chatbot Integration with Mendix That Enables 24/7 Customer Service - Watson.** (2017). Retrieved April 30, 2018, from: https://www.ibm.com/blogs/watson/2017/08/chatbot-integration-with-watson- and-mendix-enables-24x7-customer-service/
21. WOBCKE, Wayne et al**. A BDI agent architecture for dialogue modelling and coordination in a smart personal assistant**. In: Proceedings of the 2005 NICTA- HCSNet Multimodal User Interaction Workshop-Volume 57. Australian Computer Society, Inc., 2006. p. 61-66.
22. WOUDENBERG, Aswin van. **A Chatbot Dialogue Manager-Chatbots and Dialogue Systems: A Hybrid Approach**. 2014. Dissertação de Mestrado. Open Universiteit Nederland.
23. **Xcode - Apple Developer**. (2018). Retrieved April 30, 2018, from: https://developer.apple.com/xcode