A Hybrid Architecture for Multi-Party Conversational Systems

Maira Gatti de Bayser[1], Paulo Cavalin[1], Renan Souza[1], Alan Braz[1], Heloisa Candello[1], Claudio Pinhanez[1], and Jean-Pierre Briot[2]

[1]IBM Research, Rio de Janeiro, Brazil
[2]Sorbonne Universit'es, UPMC Univ Paris 06, CNRS, Laboratoire d'Informatique de Paris 6 (LIP6), Paris, France

March 7, 2024

Abstract

Multi-party Conversational Systems are systems with natural language interaction between one or more people or systems. From the moment that an utterance is sent to a group, to the moment that it is replied in the group by a member, several activities must be done by the system: utterance understanding, information search, reasoning, among others. In this paper we present the challenges of designing and building multi-party conversational systems, the state of the art, our proposed hybrid architecture using both norms and machine learning and some insights after implementing and evaluating one on the finance domain.

1 Introduction

Back to 42 BC, the philosopher Cicero has raised the issue that although there were many Oratory classes, there were none for Conversational skills [1]. He highlighted how important they were not only for politics, but also for educational purpose. Among other conversational norms, he claimed that people should be able to know when to talk in a conversation, what to talk depending on the subject of the conversation, and that they should not talk about themselves.

Norms such as these may become social conventions and are not learnt at home or at school. Social conventions are dynamic and may change according to context, culture and language. In online communication, new commonsense practices are evolved faster and accepted as a norm [2], [3]. There is not a discipline for that on elementary or high schools and there are few linguistics researchers doing research on this field.

On the other hand, within the Artificial Intelligence area, some Conversational Systems have been created in the past decades since the test proposed by Alan Turing in 1950. The test consists of a machine's ability to exhibit intelligent behavior equivalent to, or indistinguishable from that of a human [4]. Turing proposed that a human evaluator would judge natural language conversations between a human and a machine that is designed to generate human-like responses. Since then, many systems have been created to pass the

Turing's test. Some of them have won prizes, some not [5]. Although in this paper we do not focus on creating a solution that is able to build conversational systems that pass the Turing's test, we focus on Natural Dialogue Systems (NDS). From [6], "NDS *are systems that try to improve usability and user satisfaction by imitating human behavior"*. We refer to Conversational Systems as NDS, where the dialogues are expressed as natural language texts, either from artificial intelligent agents (a.k.a. bots) or from humans.

That said, the current popular name to systems that have the ability to make a conversation with humans using natural language is *Chatbot*. Chatbots are typically used in conversational systems for various practical purposes, including customer service or information acquisition. Chatbots are becoming more widely used by social media software vendors. For example, Facebook[1] recently announced that it would make Facebook Messenger (its 900-million-user messaging app by 2016), into a full-fledged platform that allows businesses to communicate with users via chatbots. Google is also building a new mobile-messaging service that uses artificial intelligence know-how and chatbot technology. In addition, according to the Wall Street Journal, there are more than 2 billion users of mobile apps. Still, people can be reluctant to install apps. So it is believed that social messaging can be a platform and chatbots may provide a new conversational interface for interacting with online services, as chatbots are easier to build and deploy than apps [7].

China seems to be the place where chatbots adoption and use is most advanced today. For example, China's popular WeChat messaging platform can take payments, scan QR codes, and integrate chatbot systems. WeChat integrates e-mail, chat, videocalls and sharing of large multimedia files. Users can book flights or hotels using a mixed, multimedia interaction with active bots. WeChat was first released in 2011 by Tecent, a Chinese online-gaming and social-media firm, and today more than 700 million people use it, being one of the most popular messaging apps in the world (The Economist 2016). WeChat has a mixture of real-live customer service agents and automated replies (Olson 2016).

Still, current existing chatbot engines do not properly handle a group chat with many users and many chatbots. This makes the chatbots considerably less social, which is a problem since there is a strong demand of having social chatbots that are able to provide different kinds of services, from traveling packages to finance advisors. This happens because there is a lack of methods and tools to design and engineer the coordination and mediation among chatbots and humans, as we present in Sections 2 and 3. In this paper, we refer to conversational systems that are able to interact with one or more people or chatbots in a multi-party chat as *Multi-Party Conversational System (*MPCS*)*. Altogether, this paper is not meant to advance the state of the art on the norms for MPCS. Instead, the main contributions of this paper are threefold:

1. We discuss the challenges of designing and building MPCS (Section 2),

2. We categorize the state of the art with works that tackle each of the challenges somehow (Section 3),

3. We present our hybrid conceptual architecture (Section 4), and insights and lessons learned after implementing and validating one on the finance domain (Section 5).

We then present some discussion and future work in the last section.

[1] http://www.facebook.com

2 Challenges on Chattering

There are plenty of challenges in conversation contexts, and even bigger ones when people and machines participate in those contexts. Conversation is a specialized form of interaction, which follows social conventions. Social interaction makes it possible to inform, context, create, ratify, refute, and ascribe, among other things, power, class, gender, ethnicity, and culture [3]. Social structures are the norms that emerge from the contact people have with others [8], for example, the communicative norms of a negotiation, taking turns in a group, the cultural identity of a person, or power relationships in a work context.

Conventions, norms and patterns from everyday real conversations are applied when designing those systems to result in adoption and match user's expectations. [9] describes implicit interactions in a framework of interactions between humans and machines. The framework is based on the theory of implicit interactions which posits that people rely on conventions of interaction to communicate queries, offers, responses, and feedback to one another. Conventions and patterns drive our expectations about interactive behaviors. This framework helps designers and developers create interactions that are more socially appropriate. According to the author, we have interfaces which are based on explicit interaction and implicit ones. The explicit are the interactions or interfaces where people rely on explicit input and output, whereas implicit interactions are the ones that occur without user awareness of the computer behavior.

Social practices and actions are essential for a conversation to take place during the turn-by-turn moments of communication. [10] highlights that a distinguishing feature of ordinary conversation is "the local, moment-by-moment management of the distribution of turns, of their size, and what gets done in them, those things being accomplished in the course of each current speaker's turn." Management of turns and subject change in each course is a situation that occurs in real life conversations based on circumstances (internal and external) to speakers in a dialogue. Nowadays, machines are not prepared to fully understand context and change the course of conversations as humans. Managing dialogues with machines is challenging, which increases even more when more than one conversational agent is part of the same conversation. Some of those challenges in the dialogue flow were addressed by [11]. According to them, we have system-initiative, user-initiative, and mixed-initiative systems.

In the first case, system-initiative systems restrict user options, asking direct questions, such as (Table 1): "What is the initial amount of investment?" Doing so, those types of systems are more successful and easier to answer to. On the other hand, user-initiative systems are the ones where users have freedom to ask what they wish. In this context, users may feel uncertain of the capabilities of the system and starting asking questions or requesting information or services which might be quite far from the system domain and understanding capacity, leading to user frustration. There is also a mixed-initiative approach, that is, a goal-oriented dialogue which users and computers participate interactively using a conversational paradigm. Challenges of this last classification are to understand interruptions, human utterances, and unclear sentences that were not always goal-oriented.

The dialog in Table 1 has the system initiative in a question and answer mode, while the one in Table 3 is a natural dialogue system where both the user and the system take the initiative. If we add another user in the chat, then we face other challenges.

In Table 4, line 4, the user U1 invites another person to the chat and the system does not reply to this utterance, nor to utterances on lines 6, 7 and 8 which are the ones when only the users (wife and husband) should reply to. On the other hand, when the couple agrees on the period and initial value of the investment (line 9), then the system S1 (at the time

Table 1: System initiative

Turn	**Utterance**
S	Which investment option do you choose?
U	Dollar
S	What is the initial amount of investment?
U	R$ 35,000
S	What is the time period of investment?
U	2 years
S	The return of investment is R$38,000 at the end.

Table 2: User initiative

Turn	**Utterance**
U	What is an investment?
S	To invest is to allocate money (or sometimes another resource, such as time) in the expectation of some benefit in the future. In finance, the expected future benefit from investment is a return.
U	What are the types of investment?
S	There are traditional and alternative investments.
U	Which investments are traditional investments?
S	Bonds, cash and real state.

the only system in the chat) replies indicating that it will invite more systems (chatbots) that are experts on this kind of pair < period, initial value >. They then join the chat and start interacting with each other. At the end, on line 17, the user U2 interacts with U1 and they agree with the certificate option. Then, the chatbot responsible for that, S3, is the only one that replies indicating how to invest.

Table 4 is one example of interactions on which the chatbots require knowledge of when to reply given the context of the dialog. In general, we acknowledge that exist four dimensions of understanding and replying to an utterance in MPCS which a chatbot that interacts in a multi-party chat group should fulfill:

1. ***What*** *is the message/utterance about?* This task means to recognize the utterance, such as the intent of the utterance, the entity making the utterance, and features of the entity, such as a time or initial value;

2. ***Who*** *should reply to the utterance?* I.e., to whom it is addressed? Should it be to a user? Or should it be to a chatbot?

3. ***How*** *the reply should be built/generated?* For example, the reply depends on an execution of an action to generate the reply, like computing a value or guiding a car.

4. ***When*** *should the reply be sent?* For instance, perhaps the reply needs to be sent within 2 minutes, 10 minutes, 1 day, or after someone/some chatbot in the chat, or before someone, some chatbot speaks, etc.

In the next section we present the state of the art and how they fullfil some of these dimensions.

Table 3: Natural Dialogue with Mixed initiative

Turn	**Utterance**
S	How can I help you?
U	I would like to invest in Dollars, is it good?
S	Oh, that would be a great choice. What would be the initial amount of investment and for how long would you keep the money invested?
U	Maybe R$35,000 in 2 years?
S	OK, um, let me see? The return of investment is R$38,000 at the end.

Table 4: MultiParty Conversation with Mixed initiative

	Turn	**Utterance**
1	S1	How can I help you?
2	U1	I would like to invest in Dollars, is it good?
3	S1	Well, not so sure. What would be the initial amount of investment and for how long would you keep the money invested?
4	U1	Actually, I am not sure. Let me invite my husband to this chat...
5	U2	<< U2 joins the chat >>
6	U1	Honey, for how long would you like to keep the money invested?
7	U2	for 2 years
8	U1	Right, so R$ 35,000 for 2 years?
9	U2	yes
10	S1	Ok, in this case I will invite two experts to simulate for you.
11	S2	<< S2 joins the chat >>
12	S3	<< S3 joins the chat >>
13	S1	Experts, can you simulate the return of investment for R$ 35,000 in 2 years?
14	S2	Sure, in the savings account the return at the end will be R$ 38,000
15	S3	Sure, in the certificate of deposit the return at the end will be R$ 38,600
16	S1	Thanks. It looks like it is better to invest in the certificate of deposit.
17	U2	Honey, lets go with the certificate.
18	U1	Ok, it seems a good idea...
19	S3	Sure, to start you can click here.

3 Conversational Systems

In this section we discuss the state of the art on conversational systems in three perspectives: types of interactions, types of architecture, and types of context reasoning. Then we present a table that consolidates and compares all of them.

ELIZA [12] was one of the first softwares created to understand natural language processing. Joseph Weizenbaum created it at the MIT in 1966 and it is well known for acting like a psychotherapist and it had only to reflect back onto patient's statements. ELIZA was created to tackle five *"fundamental technical problems"*: the identification of critical words, the discovery of a minimal context, the choice of appropriate transformations, the generation of appropriate responses to the transformation or in the absence of critical words, and the provision of an ending capacity for ELIZA scripts.

Right after ELIZA came PARRY, developed by Kenneth Colby, who is psychiatrist at Stanford University in the early 1970s. The program was written using the MLISP language (meta-lisp) on the WAITS operating system running on a DEC PDP-10 and the code is non-portable. Parts of it were written in PDP-10 assembly code and others in MLISP. There may be other parts that require other language translators. PARRY was the first system to pass the Turing test - the psychiatrists were able to make the correct identification only 48 percent of the time, which is the same as a random guessing.

A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) [13] appeared in 1995 but current version utilizes AIML, an XML language designed for creating stimulus-response chat robots [14]. A.L.I.C.E. bot has, at present, more than 40,000 categories of knowledge, whereas the original ELIZA had only about 200. The program is unable to pass the Turing test, as even the casual user will often expose its mechanistic aspects in short conversations.

Cleverbot (1997-2014) is a chatbot developed by the British AI scientist Rollo Carpenter. It passed the 2011 Turing Test at the Technique Techno-Management Festival held by the Indian Institute of Technology Guwahati. Volunteers participate in four-minute typed conversations with either Cleverbot or humans, with Cleverbot voted 59.3 per cent human, while the humans themselves were rated just 63.3 per cent human [15].

Table 5: Categories of Classical Chatbots per Interaction and Intentions

	Interactions		**Intentions**	
	Dyadic	*Coordinated*	*Goal*	*Non Goal*
ELIZA	●	❍	◗	●
PARRY	●	❍	◗	●
A.L.I.C.E	●	◗	◗	●
Cleverbot	●	◗	❍	●

Table 6: Categories of Classical Chatbots per Architectures and Context Reasoning

	Architectures			**Context Reasoning**		
	Rule	*Data*	*Hybrid*	*Rule*	*Data*	*Hybrid*
ELIZA	●	❍	❍	◗	❍	❍
PARRY	◗	●	●	❍	❍	❍
A.L.I.C.E	❍	●	●	◗	❍	❍
Cleverbot	❍	●	❍	❍	●	❍

3.1 Types of Interactions

Although most part of the research literature focuses on the dialogue of two persons, the reality of everyday life interactions shows a substantial part of multi-user conversations, such as in meetings, classes, family dinners, chats in bars and restaurants, and in almost every collaborative or competitive environment such as hospitals, schools, offices, sports teams, etc. The ability of human beings to organize, manage, and (mostly) make productive such complex interactive structures which are multi-user conversations is nothing less than remarkable. The advent of social media platforms and messaging systems such as WhatsApp in the first 15 years of the 21st century expanded our ability as a society to have asynchronous conversations in text form, from family and friends chatgroups to whole nations conversing in a highly distributed form in social media [16].

In this context, many technological advances in the early 2010s in natural language processing (spearheaded by the IBM Watson's victory in Jeopardy [17]) spurred the availability in the early 2010s of text-based chatbots in websites and apps (notably in China [18]) and spoken speech interfaces such as Siri by Apple, Cortana by Microsoft, Alexa by Amazon, and Allo by Google. However, the absolute majority of those chatbot deployments were in contexts of dyadic dialog, that is, a conversation between a single chatbot with a single user. Most of the first toolkits for chatbot design and development of this initial period implicit assume that an utterance from the user is followed by an utterance of the chatbot, which greatly simplifies the management of the conversation as discussed in more details later. Therefore, from the interaction point of view, there are two types: 1) one in which the chatbot was designed to chat with one person or chatbot, and 2) other in which the chatbot can interact with more than two members in the chat.

Dyadic Chatbot

A Dyadic Chatbot is a chatbot that does know **when** to talk. If it receives an utterance, it will always handle and try to reply to the received utterance. For this chatbot to behave properly, either there are only two members in the chat, and the chatbot is one of them, or there are more, but the chatbot replies only when its name or nickname is mentioned. This means that a dyadic chatbot does not know how to coordinate with many members in a chat group. It lacks the social ability of knowing when it is more suitable to answer or not. Also, note that we are not considering here the ones that would use this social ability as an advantage in the conversation, because if the chatbot is doing with this intention, it means that the chatbot was designed to be aware of the social issues regarding a chat with multiple members, which is not the case of a dyadic chatbot. Most existing chatbots, from the first system, ELIZA [12], until modern state-of-the-art ones fall into this category.

Multiparty Conversations

In multiparty conversations between people and computer systems, natural language becomes the communication protocol exchanged not only by the human users, but also among the bots themselves. When every actor, computer or user, understands human language and is able to engage effectively in a conversation, a new, universal computer protocol of communication is feasible, and one for which people are extremely good at.

There are many differences between dyadic and multiparty conversations, but chiefly among them is turn-taking, that is, how a participant determines when it is appropriate to make an utterance and how that is accomplished. There are many social settings, such as assemblies, debates, one-channel radio communications, and some formal meetings, where there are clear and explicit norms of who, when, and for long a participant can speak.

The state of the art for the creation of chatbots that can participate on multiparty conversations currently is a combination of the research on the creation of chatbots and research on the coordination or governance of multi-agents systems. A definition that mixes both concepts herein present is: *A **chatbot** is an **agent** that interacts through natural language.* Although these areas complement each other, there is a lack of solutions for creating multiparty-aware chatbots or governed chatbots, which can lead to higher degree of system trust.

(i) *Multi-Dyadic Chatbots*

Turn-taking in generic, multiparty spoken conversation has been studied by, for example, Sacks et al. [19]. In broad terms, it was found that participants in general do not overlap their utterances and that the structure of the language and the norms of conversation create specific moments, called transition-relevance places, where turns can occur. In many cases, the last utterances make clear to the participants who should be the next speaker (selected-next-speaker), and he or she can take that moment to start to talk. Otherwise, any other participant can start speaking, with preference for the first starter to get the turn; or the current speaker can continue [19].

A key part of the challenge is to determine whether the context of the conversation so far have or have not determined the next speaker. In its simplest form, a vocative such as the name of the next speaker is uttered. Also, there is a strong bias towards the speaker before the current being the most likely candidate to be the next speaker.

In general the detection of transition-relevance places and of the selected-next-speaker is still a challenge for speech-based machine conversational systems. However, in the case of text message chats, transition-relevance places are often determined by the acting of posting a message, so the main problem facing multiparty-enabled textual chatbots is in fact determining whether there is and who is the selected-next-speaker. In other words, *chatbots have to know when to shut up*. Bohus and Horowitz [20] have proposed a computational probabilistic model for speech-based systems, but we are not aware of any work dealing with modeling turn-taking in textual chats.

(ii) *Coordination of Multi-Agent Systems*

A multi-agent system (MAS) can be defined as a computational environment in which individual software agents interact with each other, in a cooperative manner, or in a competitive manner, and sometimes autonomously pursuing their individual goals. During this process, they access the environment's resources and services and occasionally produce results for the entities that initiated these software agents. As the agents interact in a concurrent, asynchronous and decentralized manner, this kind of system can be categorized as a complex system [21].

Research in the coordination of multi-agent systems area does not address coordination using natural dialogue, as usually all messages are structured and formalized so the agents can reason and coordinate themselves. On the other hand, chatbots coordination have some relations with general coordination mechanisms of multi-agent systems in that they specify and control interactions between agents. However, chatbots coordination mechanisms is meant to regulate interactions and actions from a social perspective, whereas general coordination languages and mechanisms focus on means for expressing synchronization and coordination of activities and exchange of information, at a lower computational level.

In open multi-agent systems the development takes place without a centralized control, thus it is necessary to ensure the reliability of these systems in a way that all the interactions between agents will occur according to the specification and that these agents will obey the specified scenario. For this, these applications must be built upon a law-governed architecture.

Minsky published the first ideas about laws in 1987 [22]. Considering that a law is a set of norms that govern the interaction, afterwards, he published a seminal paper with the Law-Governed Interaction (LGI) conceptual model about the role of interaction laws on distributed systems [23]. Since then, he conducted further work and experimentation based on those ideas [24]. Although at the low level a multiparty conversation system is a distributed system and the LGI conceptual model can be used in a variety of application domains, it is composed of abstractions basically related to low level information about communication issues of distributed systems (like the primitives *disconnected*, *reconnected*, *forward*, and *sending* or *receiving of messages*), lacking the ability to express high level information of social systems.

Following the same approach, the Electronic Institution (EI) [25] solution also provides support for interaction norms. An EI has a set of high-level abstractions that allow for the specification of laws using concepts such as agent roles, norms and scenes.

Still at the agent level but more at the social level, the XMLaw description language and the M-Law framework [26] [27] were proposed and developed to support law-governed mechanism. They implement a law enforcement approach as an object-oriented framework and it allows normative behavior through the combination between norms and clocks. The M-Law framework [27] works by intercepting messages exchanged between agents, verifying the compliance of the messages with the laws and subsequently redirecting the message to the real addressee, if the laws allow it. If the message is not compliant, then the mediator blocks the message and applies the consequences specified in the law, if any. They are called laws in the sense that they **enforce** the norms, which represent what can be done (permissions), what cannot be done (prohibitions) and what must be done (obligations).

(iii) *Coordinated Aware Chatbots in a Multiparty Conversation*

With regard to chatbot engines, there is a lack of research directed to building coordination laws integrated with natural language. To the best of our knowledge, the architecture proposed in this paper is the first one in the state of the art designed to support the design and development of coordinated aware chatbots in a multiparty conversation.

3.2 Types of Architectures

There are mainly three types of architectures when building conversational systems: totally rule-oriented, totally data-oriented, and a mix of rules and data-oriented.

Rule-oriented

A rule-oriented architecture provides a manually coded reply for each recognized utterance. Classical examples of rule-based chatbots include Eliza and Parry. Eliza could also extract some words from sentences and then create another sentence with these words based on their syntatic functions. It was a rule-based solution with no reasoning. Eliza could not "understand" what she was parsing. More sophisticated rule-oriented architectures contain grammars and mappings for converting sentences to appropriate sentences using some sort of knowledge. They can be implemented with propositional logic or first-order logic (FOL). Propositional logic assumes the world contains facts (which refer to events, phenomena, symptoms or activities). Usually, a set of facts (statements) is not sufficient to describe a domain in a complete manner. On the other hand, FOL assumes the world contains Objects (e.g., people, houses, numbers, etc.), Relations (e.g. red, prime, brother of, part of, comes between, etc.), and Functions (e.g. father of, best friend, etc.), not only facts as in propositional logic. Moreover, FOL contains predicates, quantifiers and variables, which range over individuals (which are domain of discourse).

Prolog (from French: *Programmation en Logique*) was one of the first logic programming languages (created in the 1970s), and it is one of the most important languages for expressing phrases, rules and facts. A Prolog program consists of logical formulas and running a program means proving a theorem. Knowledge bases, which include rules in addition to facts, are the basis for most rule-oriented chatbots created so far.

In general, a rule is presented as follows:

$$if < premise > then < conclusion > \tag{1}$$

Prolog made it possible to perform the language of Horn clauses (implications with only one conclusion). The concept of Prolog is based on predicate logic, and proving theorems involves a resolute system of denials. Prolog can be distinguished from classic programming languages due to its possibility of interpreting the code in both a procedural and declarative way. Although Prolog is a set of specifications in FOL, it adopts the closed-world assumption, i.e. all knowledge of the world is present in the database. If a term is not in the database, Prolog assumes it is false.

In case of Prolog, the FOL-based set of specifications (formulas) together with the facts compose the knowledge base to be used by a rule-oriented chatbot. However an Ontology could be used. For instance, OntBot [28] uses mapping technique to transform ontologies and knowledge into relational database and then use that knowledge to drive its chats. One of the main issues currently facing such a huge amount of ontologies stored in a database is the lack of easy to use interfaces for data retrieval, due to the need to use special query languages or applications.

In rule-oriented chatbots, the degree of intelligent behavior depends on the knowledge base size and quality (which represents the information that the chatbot knows), poor ones lead to weak chatbot responses while good ones do the opposite. However, good knowledge bases may require years to be created, depending on the domain.

Data-oriented

As opposed to rule-oriented architectures, where rules have to be explicitly defined, data-oriented architectures are based on learning models from samples of dialogues, in order to reproduce the behavior of the interaction that are observed in the data. Such kind of learning can be done by means of machine learning approach, or by simply extracting rules from data instead of manually coding them.

Among the different technologies on which these system can be based, we can highlight classical information retrieval algorithms, neural networks [29], Hidden Markov Models (HMM) [30], and Partially Observable Markov Decision Process (POMDP) [31]. Examples include Cleverbot and Tay [32]. Tay was a chatbot developed by Microsoft that after one day live learning from interaction with teenagers on Twitter, started replying impolite utterances. Microsoft has developed others similar chatbots in China (Xiaoice[2]) and in Japan (Rinna[3]). Microsoft has not associated its publications with these chatbots, but they have published a data-oriented approach[33] that proposes a unified multi-turn multi-task spoken language understanding (SLU) solution capable of handling multiple context sensitive classification (intent determination) and sequence labeling (slot filling) tasks simultaneously. The proposed architecture is based on recurrent convolutional neural networks (RCNN) with shared feature layers and globally normalized sequence modeling components.

A survey of public available corpora for can be found in [34]. A corpus can be classified into different categories, according to: the type of data, whether it is spoken dialogues, transcripts of spoken dialogues, or directly written; the type of interaction, if it is human-human or human-machine; and the domain, whether it is restricted or unconstrained. Two well-known corpora are the Switchboard dataset, which consists of transcripts of spoken, unconstrained, dialogues, and the set of tasks for the Dialog State Tracking Challenge (DSTC), which contain more constrained tasks, for instance the restaurant and travel information sets.

Rule and Data-oriented

The model of learning in current A.L.I.C.E. [14] is incremental or/and interactive learning because a person monitors the robot's conversations and creates new AIML content to make the responses more appropriate, accurate, believable, "human", or whatever he/she intends. There are algorithms for automatic detection of patterns in the dialogue data and this process provides the person with new input patterns that do not have specific replies yet, permitting a process of almost continuous supervised refinement of the bot.

As already mentioned, A.L.I.C.E. consists of roughly 41,000 elements called categories which is the basic unit of knowledge in AIML. Each category consists of an input question, an output answer, and an optional context. The question, or stimulus, is called the pattern. The answer, or response, is called the template. The two types of optional context are called *that* and *topic*. The keyword *that* refers to the robot's previous utterance. The AIML pattern language consists only of words, spaces, and the wildcard symbols "_" and "*". The words may consist only of letters and numerals. The pattern language is case invariant. Words are separated by a single space, and the wildcard characters function like words, similar to the initial pattern matching strategy of the Eliza system. More generally, AIML tags transform the reply into a mini computer program which can save data, activate other programs, give conditional responses, and recursively call the pattern matcher to insert the responses from other categories. Most AIML tags in fact belong to this template side sublanguage [14].

AIML language allows:

1. Symbolic reduction: Reduce complex grammatical forms to simpler ones.

[2] http://www.msxiaoice.com/

[3] http://rinna.jp/

2. Divide and conquer: Split an input into two or more subparts, and combine the responses to each.

3. Synonyms: Map different ways of saying the same thing to the same reply.

4. Spelling or grammar corrections: the bot both corrects the client input and acts as a language tutor.

5. Detecting keywords anywhere in the input that act like triggers for a reply.

6. Conditionals: Certain forms of branching to produce a reply.

7. Any combination of (1)-(6).

When the bot chats with multiple clients, the predicates are stored relative to each client ID. For example, the markup *<set name="name">Matthew</set >* stores the string *Matthew* under the predicate named *"name"*. Subsequent activations of *<get name="name">* return *"Matthew"*. In addition, one of the simple tricks that makes ELIZA and A.L.I.C.E. so believable is a pronoun swapping substitution. For instance:

U: My husband would like to invest with me.
S: Who else in your family would like to invest with you?

3.3 Types of Intentions

According to the types of intentions, conversational systems can be classified into two categories: a) goal-driven or task oriented, and b) non-goal-driven or end-to-end systems.

In a goal-driven system, the main objective is to interact with the user so that back-end tasks, which are application specific, are executed by a supporting system. As an example of application we can cite technical support systems, for instance air ticket booking systems, where the conversation system must interact with the user until all the required information is known, such as origin, destination, departure date and return date, and the supporting system must book the ticket. The most widely used approaches for developing these systems are Partially-observed Decision Processes (POMDP) [31], Hidden Markov Models (HMM) [30], and more recently, Memory Networks [29]. Given that these approaches are data-oriented, a major issue is to collect a large corpora of annotated task-specific dialogs. For this reason, it is not trivial to transfer the knowledge from one to domain to another. In addition, it might be difficult to scale up to larger sets of tasks.

Non-goal-driven systems (also sometimes called reactive systems), on the other hand, generate utterances in accordance to user input, e.g. language learning tools or computer games characters. These systems have become more popular in recent years, mainly owning to the increase of popularity of Neural Networks, which is also a data-oriented approach. The most recent state of the art to develop such systems have employed Recurrent Neural Networs (RNN) [35], Dynamic Context-Sensitive Generation [36], and Memory Networks [37], just to name a few. Nevertheless, probabilistic methods such as Hidden Topic Markov Models (HTMM) [38] have also been evaluated. Goal-driven approach can create both proactive and reactive chatbots, while non-goal-driven approach creates reactive chatbots. In addition, they can serve as a tool to goal-driven systems as in [29]. That is, when trained on corpora of a goal-driven system, non-goal-driven systems can be used to simulate user interaction to then train goal-driven models.

3.4 Types of Context Reasoning

A dialogue system may support the context reasoning or not. Context reasoning is necessary in many occasions. For instance, when partial information is provided the chatbot needs to be able to interact one or more turns in order to get the complete information in order to be able to properly answer. In [39], the authors present a taxonomy of errors in conversational systems. The ones regarding context-level errors are the ones that are perceived as the top-10 confusing and they are mainly divided into the following:

- Excess/lack of proposition: the utterance does not provide any new proposition to the discourse context or provides excessive information than required.
- Contradiction: the utterance contains propositions that contradict what has been said by the system or by the user.
- Non-relevant topic: the topic of the utterance is irrelevant to the current context such as when the system suddenly jumps to some other topic triggered by some particular word in the previous user utterance.
- Unclear relation: although the utterance might relate to the previous user utterance, its relation to the current topic is unclear.
- Topic switch error: the utterance displays the fact that the system missed the switch in topic by the user, continuing with the previous topic.

Rule-oriented

In the state of the art most of the proposed approaches for context reasoning lies on rules using logics and knowledge bases as described in the Rule-oriented architecture sub-section. Given a set of facts extracted from the dialogue history and encoded in, for instance, FOL statements, a queries can be posed to the inference engine and produce answers. For instance, see the example in Table 7. The sentences were extracted from [37] (which does not use a rule-oriented approach), and the first five statements are their respective facts. The system then apply context reasoning for the query *Q: Where is the apple.*

Table 7: Example of sequence of utterances and FOL statements

Utterance	**FOL statement**
Sam walks into the kitchen.	$isAt(Sam, kitchen)$
Sam picks up an apple.	$pickUp(Sam, apple)$
Sam walks into the bedroom.	$isAt(Sam, bedroom)$
Sam drops the apple.	$\forall x(\forall y(\forall w(drops(x, y) \wedge isAt(x, w) \rightarrow isAt(y, w))))$
Q: Where is the apple?	
A: Bedroom	isAt(apple, bedroom)

If statements above are received on the order present in Table 7, if the query *Q: Where is the apple* is sent, the inference engine will produce the answer *A: Bedroom* (i.e., the statement $isAt(apple, bedroom)$ is found by the model and returned as True).

Nowadays, the most common way to store knowledge bases is on triple stores, or RDF (Resource Description Framework)[4] stores. A triple store is a knowledge base for the storage and retrieval of triples through semantic queries. A triple is a data entity composed of subject-predicate-object, like "Sam is at the kitchen" or "The apple is with Sam", for instance. A query language is needed for storing and retrieving data from a triple store. While SPARQL is a RDF query language, Rya[5] is an open source scalable RDF triple store built on top of Apache Accumulo[6]. Originally developed by the Laboratory for Telecommunication Sciences and US Naval Academy, Rya is currently being used by a number of american government agencies for storing, inferencing, and querying large amounts of RDF data.

A SPARQL query has a SQL-like syntax for finding triples matching specific patterns. For instance, see the query below. It retrieves all the people that works at IBM and lives in New York:

```
SELECT ?people
WHERE {
?people <worksAt> <IBM> .
?people <livesIn> <New York>.
}
```

Since triple stores can become huge, Rya provides three triple table index [40] to help speeding up queries:

SPO: subject, predicate, object
POS: predicate, object, subject
OSP: object, subject, predicate

While Rya is an example of an optimized triple store, a rule-oriented chatbot can make use of Rya or any triple store and can call the semantic search engine in order to inference and generate proper answers.

Data-oriented

Recent papers have used neural networks to predict the next utterance on non-goal-driven systems considering the context, for instance with Memory Networks [41]. In this work [37], for example the authors were able to generate answers for dialogue like below:

```
Sam walks into the kitchen.
Sam picks up an apple.
Sam walks into the bedroom.
Sam drops the apple.
Q: Where is the apple?
A: Bedroom
```

[4] The Resource Description Framework (RDF) is a standard model for expressing graph data for the World Wide Web.

[5] https://rya.apache.org/

[6] https://accumulo.apache.org/

Sukhbaatar's model represents the sentence as a vector in a way that the order of the words matter, and the model encodes the temporal context enhancing the memory vector with a matrix that contains the temporal information. During the execution phase, Sukhbaatar's model takes a discrete set of inputs $x_1, ..., x_n$ that are to be stored in the memory, a query q, and outputs an answer a. Each of the x_i, q, and a contains symbols coming from a dictionary with V words. The model writes all x to the memory up to a fixed buffer size, and then finds a continuous representation for the x and q. The continuous representation is then processed via multiple computational steps to output a. This allows back propagation of the error signal through multiple memory accesses back to the input during training. Sukhbaatar's also presents the state of the art of recent efforts that have explored ways to capture dialogue context, treated as long-term structure within sequences, using RNNs or LSTM-based models. The problem of this approach is that it is has not been tested for goal-oriented systems. In addition, it works with a set of sentences but not necessary from multi-party bots.

3.5 Platforms

Regarding current platforms to support the development of conversational systems, we can categorize them into three types: platforms for plugging chatbots, for creating chatbots and for creating service chatbots. The platforms for plugging chatbots provide tools for integrating them another system, like Slack[7]. The chatbots need to receive and send messages in a specific way, which depends on the API and there is no support for actually helping on building chatbots behavior with natural language understanding. The platforms for creating chatbots mainly provide tools for adding and training intentions together with dialogue flow specification and some entities extraction, with no reasoning support. Once the models are trained and the dialogue flow specified, the chatbots are able to reply to the received intention. The platforms for creating service chatbots provide the same functionalities as the last one and also provide support for defining actions to be executed by the chatbots when they are answering to an utterance. Table 8 summarizes current platforms on the market accordingly to these categories. There is a lack on platforms that allow to create chatbots that can be coordinated in a multiparty chat with governance or mediation.

Table 8: Platforms for Building Chatbots

	Plugging Bots	**Creating Bots**	**Creating Service Bots**	**Coordination Control**
IBM (Watson)	❍	●	◗	❍
Pandora	❍	●	●	❍
Facebook (Wit.ai)	❍	●	●	❍
Microsoft (LUIS)	❍	●	❍	❍
Google Hangout	●	❍	❍	❍
Slack	●	❍	❍	❍
WeChat	●	●	●	❍
Kik	●	❍	❍	❍

[7] www.slack.com

4 A Conceptual Architecture for Multiparty-Aware Chatbots

In this section the conceptual architecture for creating a hybrid rule and machine learning-based MPCS is presented. The MPCS is defined by the the entities and relationships illustrated in Fig. 1 which represents the chatbot's knowledge. A Chat Group contains several Members that join the group with a Role. The role may constrain the behavior of the member in the group. Chatbot is a type of Role, to differentiate from persons that may also join with different roles. For instance, a person may assume the role of the owner of the group, or someone that was invited by the owner, or a domain role like an expert, teacher or other.

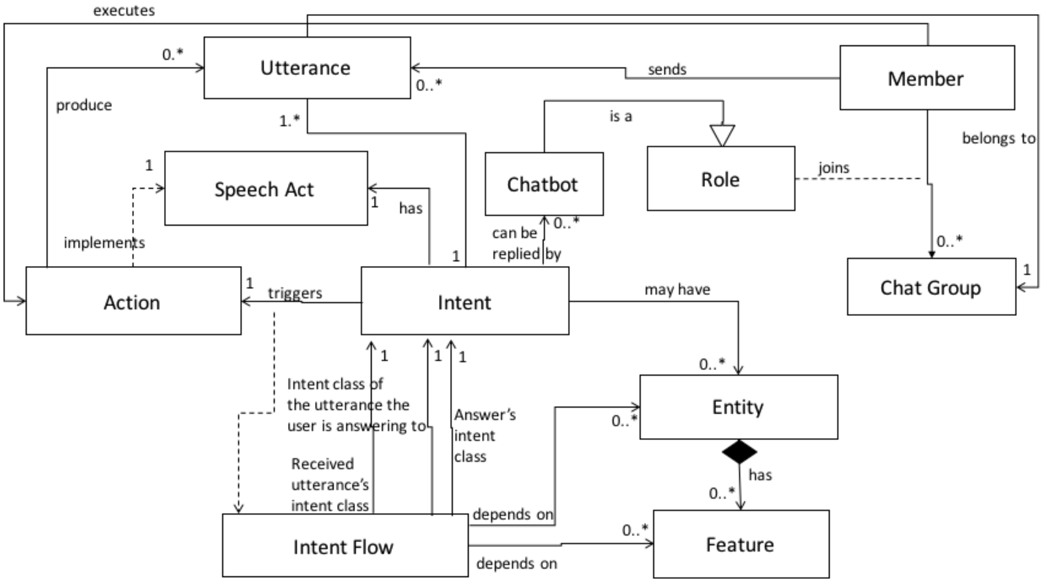

Figure 1: Chatbot Knowledge's Conceptual Model in a MPCS

When a Member joins the Chat Group, it/he/she can send Utterances. The Member then classifies each Utterance with an Intent which has a Speech Act[8]. The Intent class, Speech Act class and the Intent Flow trigger the Action class to be executed by the Member that is a Chatbot. The Chatbots associated to the Intention are the only ones that know how to answer to it by executing Actions. The Action, which implements one Speech Act, produces answers which are Utterances, so, for instance, the Get_News action produces an Utterance for which Intention's speech act is Inform_News. The Intent Flow holds the intent's class conversation graph which maps the dialog state as a decision tree. The answer's intention class is mapped in the Intent Flow as a directed graph G defined as following:

$$G = (V, R)|(x, y) = (< I_u, I_r >, I_a) \tag{2}$$

[8] Or dialogue act, or communicative act, of performative, depending on the research community.

From the graph definitions, V is for *vertices* and R is for *relations*, which are the arrows in the graph. And in Equation 2: V is the set of intentions pairs,
R is the set of paths to navigate through the intentions,
$< I_u, I_r >$ is the arrow's head, and
I_a is the arrow's tail.

This arrow represents a turn from an utterance with I_u intention class which is replying to an utterance with I_r intention class to the state which an utterance with I_r intention's class is sent.

I_a is the intention class of the answer to be provided to the received I_u intention class.

In addition, each intent's class may refer to many `Entities` which, in turn, may be associated to several `Features`. For instance, the utterance

"I would like to invest USD10,000 in Savings Account for 2 years"

contains one entity – the Savings Account's investment option – and two features – money (USD10,000) and period of time (2 years). The `Intent Flow` may need this information to choose the next node which will give the next answer. Therefore, if the example is changed a little, like

"I would like to invest in Savings Account",

I_a is constrained by the "Savings Account" entity which requires the two aforementioned features. Hence, a possible answer by one `Member` of the group would be

"Sure, I can simulate for you, what would be the initial amount and the period of time of the investment?"

With these conceptual model's elements, a MPCS system can be built with multiple chatbots. Next subsection further describes the components workflow.

4.1 Workflow

Figure 2 illustrates from the moment that an utterance is sent in a chat group to the moment a reply is generated in the same chat group, if the case. One or more person may be in the chat, while one or more chatbots too. There is a `Hub` that is responsible for broadcasting the messages to every `Member` in the group, if the case. The flow starts when a `Member` sends the utterance which goes to the `Hub` and, if allowed, is broadcasted. Many or none interactions norms can be enforced at this level depending on the application. Herein, a norm can be a prohibition, obligation or permission to send an utterance in the chat group.

Once the utterance is broadcasted, a chatbot needs to handle the utterance. In order to properly handle it, the chatbot parses the utterance with several parsers in the `Parsing phase`: a `Topic Classifier`, the `Dependency Parsing`, which includes Part-of-Speech tags and semantics tags, and any other that can extract metadata from the utterance useful for the reasoning. All these metadata, together with more criteria, may be used in the `Frame parsing` which is useful for context reasoning. All knowledge generated in this phase can be stored in the `Context`. Then, the `Intent Classifier` tries to detect the intent class of the utterance. If detected, the `Speech Act` is also retrieved. And an `Event Detector` can also check if there is any dialog inconsistency during this phase.

After that, the `Filtering phase` receives the object containing the utterance, the detected intent, and all metadata extracted so far and decides if an action should be performed to reply to the utterance. If yes, it is sent to the `Acting phase` which performs several steps. First the `Action Classifier` tries to detect the action to be performed. If detected, the action is executed. At this step, many substeps may be performed, like searching for an information, computing maths, or generating information to create the answer. All of this may require a search in the `Context` and also may activate the `Error Detector` component to check if the dialog did not run into a wrong state. After the answer is generated, the `Filtering phase` is activated again to check if the reply should be really sent. If so, it is sent to the `Hub` which, again may check if it can be broadcasted before actually doing it.

4.1.1 Topic Classifier

The topic classifier is domain-dependent and is not mandatory. However, the chatbot can better react when the intent or action is not detected, which means that it does not know how to answer. Many reasons might explain this situation: the set of intents might be incomplete, the action might not have produced the proper behavior, misunderstanding

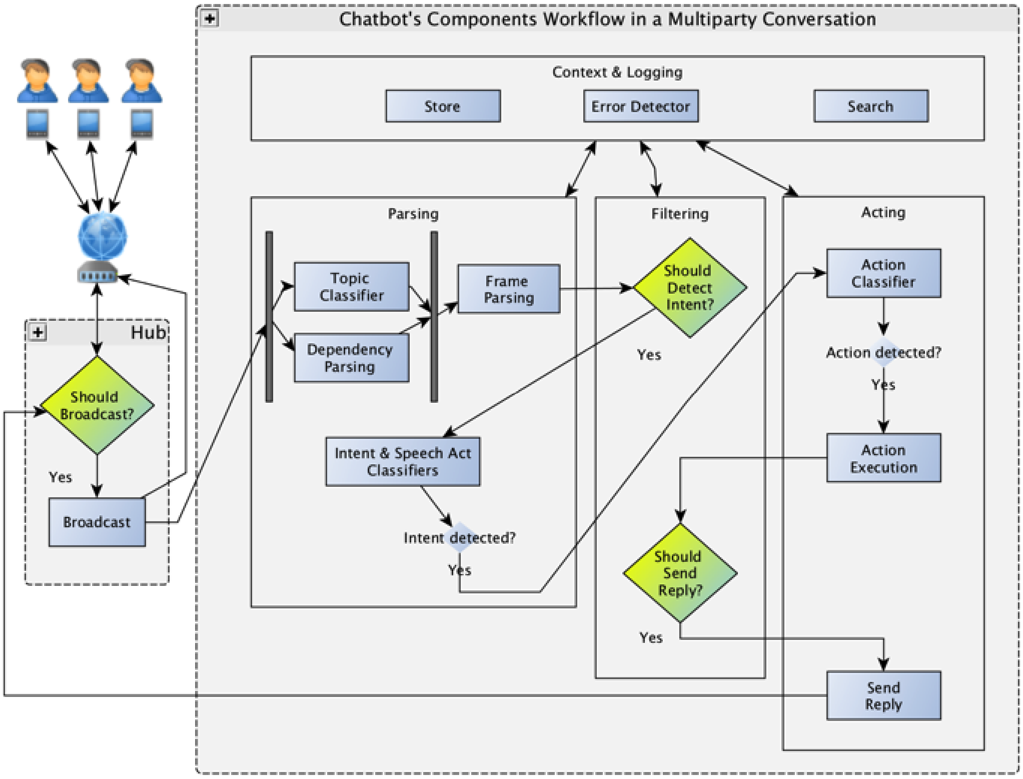

Figure 2: Workflow

might happen, or the chatbot was not designed to reply to a particular topic. In all cases, it must be able to produce a proper reply, if needed. Because this might happen throughout the workflow, the sooner that information is available, the better the chatbot reacts. Therefore it is one of the first executions of the flow.

4.1.2 Dependency Parsing

Dependency is the notion that linguistic units, e.g. words, are connected to each other by directed links. The (finite) verb is taken to be the structural center of clause structure. All other syntactic units (words) are either directly or indirectly connected to the verb in terms of the directed links, which are called dependencies. It is a one-to-one correspondence: for every element (e.g. word or morph) in the sentence, there is exactly one node in the structure of that sentence that corresponds to that element. The result of this one-to-one correspondence is that dependency grammars are word (or morph) grammars. All that exist are the elements and the dependencies that connect the elements into a structure. Dependency grammar (DG) is a class of modern syntactic theories that are all based on the dependency relation.

Semantic dependencies are understood in terms of predicates and their arguments. Morphological dependencies obtain between words or parts of words. To facilitate future research in unsupervised induction of syntactic structure and to standardize best-practices, a tagset that consists of twelve universal part-of-speech categories was proposed [42].

Dependency parsers have to cope with a high degree of ambiguity and nondeterminism which let to different techniques than the ones used for parsing well-defined formal languages. Currently the mainstream approach uses algorithms that derive a potentially very large set of analyses in parallel and when disambiguation is required, this approach can be coupled with a statistical model for parse selection that ranks competing analyses with respect to plausibility [43].

Below we present an example of a dependency tree for the utterance:

"I want to invest 10 thousands":

```
"tree": {
    "want␣VERB␣ROOT": {
      "I␣PRON␣nsubj": {},
      "to␣ADP␣mark": {},
      "invest␣VERB␣nmod": {
          "thousands␣NOUN␣nmod": {
               "10␣NUM␣nummod": {}
          }
      }
    }
```

The coarse-grained part-of-speech tags, or morphological dependencies (`VERB`, `PRON`, `ADP`, `NOUN` and `NUM`) encode basic grammatical categories and the grammatical relationships (`nsubjs`, `nmod`, `nummod`) are defined in the Universal Dependencies project[9] [42].

[9] http://universaldependencies.org/

4.1.3 Frame Parsing

In this module, the dependency tree generated is used together with a set of rules to extract information that is saved in the context using the Frame-based approach. This approach fills the slots of the *frame* with the extracted values from the dialogue. Frames are like forms and slots are like fields. Using the knowledge's conceptual model, the fields are represented by the elements `Entities` and `Features`. In the dependency tree example, the entity would be the implicit concept: the investment option, and the feature is the implicit concept: initial amount – `10 thousands`. Since the goal is `to invest`, and there are more entities needed for that (i.e., fields to be filled), the next node in the Intention Flow tree would return an utterance which asks the user the time of investment, if he/she has not provided yet.

This module could be implemented using different approaches according to the domain, but tree search algorithms will be necessary for doing the tree parsing.

4.1.4 Intent Classifier

The `Intent Classifier` component aims at recognizing not only the `Intent` but the *goal* of the utterance sent by a `Member`, so it can properly react. The development of an intent classifier needs to deal with the following steps:

i) the creation of dataset of intents, to train the classification algorithm;
ii) the design of a classification algorithm that provides a reasonable level of accuracy;
iii) the creation of dataset of trees of intents, the same as defined in i) and which maps the goals;
iv) the design of a plan-graph search algorithm that maps the goal's state to a node in the graph;

There are several approaches to create training sets for dialogues: from an incremental approach to crowdsourcing. In the incremental approach, the Wizard of Oz method can be applied to a set of potential users of the system, and from this study, a set of questions that the users asked posted to the 'fake' system can be collected. These questions have to be manually classified into a set of intent classes, and used to train the first version of the system. Next, this set has to be increased both in terms of number of classes and samples per class.

4.1.5 Speech Act Classifier

The `Speech Act Classifier` can be implemented with many speech act classes as needed by the application. The more classes, the more flexible the chatbot is. It can be built based on dictionaries, or a machine learning-based classifier can be trained. In the table below we present the main and more general speech act classes [44] used in the Chatbots with examples to differentiate one from another:

4.1.6 Action Classifier

There are at least as many `Action` classes as `Speech Act` classes, since the action is the realization of a speech act. The domain specific classes, like `"Inform_News"` or `"Inform_Factoids"`, enhance the capabilities of answering of a chatbot.

The `Action Classifier` can be defined as a multi-class classifier with the tuple

Table 9: Main Generic Speech Acts Classes

Speech Act	**Example**
GREETINGS	*Hello*
THANK	*Thank you very much*
INFORM	*The stock market is launched*
QUERY	*Is the stock market launched?*
CFP	*Can someone launch the stock market?*
REQUEST	*Launch the stock market*
AGREE	*OK, I will launch the stock market*
REFUSE	*I will not launch the stock market*
FAILURE	*I can not launch the stock market*
PROPOSE	*I can launch the stock market for X dollars*
SUBSCRIBE	*I would like to know when someone launches the stock market*
NOT-UNDERSTOOD	*Stock market? Which stock market?*
BYE	*See you*

$$A =< I_a, S_a, E, F > \tag{3}$$

where I_a is the intent of the answer defined in (2), S_a is the speech act of the answer, E and F are the sets of entities and features needed to produce the answer, if needed, respectively.

4.1.7 Action Execution

This component is responsible for implementing the behavior of the `Action` class. Basic behaviors may exist and be shared among different chatbots, like the ones that implement the greetings, thanks or not understood. Although they can be generic, they can also be personalized to differentiate the bot from one another and also to make it more "real". Other cases like to inform, to send a query, to send a proposal, they are all domain-dependent and may require specific implementations.

Anyway, figure 3 shows at the high level the generic workflow. If action class detected is task-oriented, the system will implement the execution of the task, say to guide a car, to move a robot's arm, or to compute the return of investments. The execution might need to access an external service in the Internet in order to complete the task, like getting the inflation rate, or the interest rate, or to get information about the environment, or any external factor. During the execution or after it is finished, the utterance is generated as a reply and, if no more tasks are needed, the action execution is finished.

In the case of coordination of chatbots, one or more chatbots with the role of mediator may exist in the chat group and, at this step, it is able to invite one or more chatbots to the chat group and it is also able to redirect the utterances, if the case.

The proposed architecture addresses the challenges as the following:

1. ***What** is the message/utterance about?* solved by the ***Parsing phase***;
2. ***Who** should reply to the utterance?* solved by the ***Filtering phase*** and may be enforced by the ***Hub***;

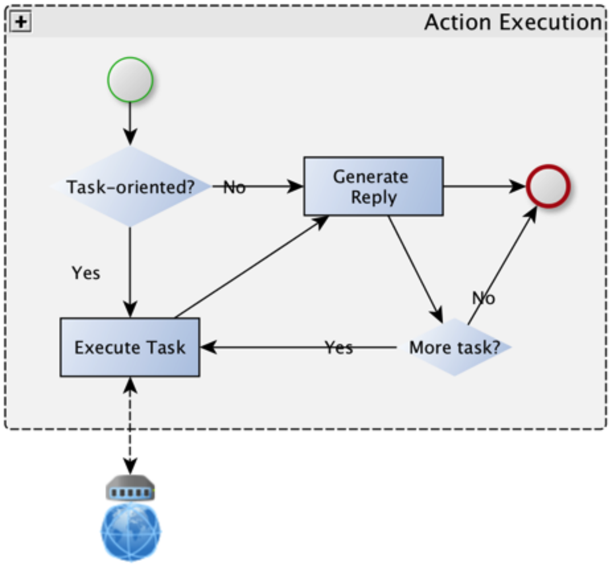

Figure 3: Action Execution

3. ***How** the reply should be built/generated?* solved by the ***Acting phase***;

4. ***When** should the reply be sent?* may be solved by the ***Acting phase*** or the ***Filtering phase***, and may be enforced by the ***Hub***;

And `Context and Logging` module is used throughout all phases.

5 Architecture Implementation and Evaluation

This section presents one implementation of the conceptual architecture presented in last section. After many refactorings, a framework called SABIA (Speech-Act-Based Intelligent Agents Framework) has been developed and CognIA (Cognitive Investment Advisor) application has been developed as an instantiation of SABIA framework. We present then the accuracy and some automated tests of this implementation.

5.1 Speech-Act-based Intelligent Agents Framework

SABIA was developed on top of Akka middleware[10]. Akka is a toolkit and runtime that implements the Actor Model on the JVM. Akka's features, like concurrency, distributed computing, resilience, and message-passing were inspired by Erlang's actor model [45] [46]. The actor model is a mathematical model of concurrent computation that treats "actors" as the universal primitives of concurrent computation. In response to a message that it receives, an actor can: make local decisions, create more actors, send more messages, and determine how to respond to the next received message. Actors may modify private state,

[10] http://akka.io

but can only affect each other through messages (avoiding the need for any locks). Akka middleware manages the actors life cycle and actors look up by theirs name, locally or remotely.

We implemented each Member of the Chat Group as an Actor by extending the UntypedActor class of Akka middleware. Yet, we created and implemented the SabiaActorSystem as a singleton (i.e., a single instance of it exists in the system) [47] that has a reference to Akka's ActorSystem. During SabiaActorSystem's initialization, all parsers that consume too much memory during their initialization to load models are instantiated as singletons. In this way, we save time on their calls during the runtime. Moreover, all chat group management, like to join or leave the group, or to broadcast or filter a message at the Hub level is implemented in SABIA through the Chat Group behavior.

5.1.1 Dependency Parsing

This is implemented in SABIA as a singleton that is initialized during the SabiaActorSystem initialization with the URL of the service that implements the dependency parsing and is used on each utterance's arrival through the execution of the tagUtterance method. The service must retrieve a *JSON Object* with the dependency tree which is then parsed using depth-first search.

5.1.2 Frame Parsing

SABIA does not support invariants for frame parsing. We are leaving this task to the instantiated application.

5.1.3 Intent Classifier

There are two intent classifiers that can be loaded with trained models in order to be ready to be used at runtime: the 1-nearest-neighbor (1NN) and the SVM-based classifier.

5.1.4 Action Classifier

SABIA implements the Action Classifier assuming that the application uses a relational database with a data schema that implements the conceptual model presented in Figure 1. Then the invariants parts that use SQL are already present and the application only needs to implement the database connection and follow the required data schema.

5.1.5 Action Execution

SABIA provides partial implemented behavior for the Action through the Template method design pattern [47], which implements the invariants parts of the action execution and leaves placeholders for customization.

5.2 CognIA: A Cognitive Investment Advisor

We developed CognIA, which is an instantiation of Sabia framework. A conversation is composed of a group chat that can contain multiple users and multiple chatbots. This example, in particular, has a mediator that can help users on financial matters, more specifically on investment options. For example, consider the following dialogue in the table below:

Table 10: MultiParty Conversation with Mixed initiative

	Turn	**Utterance**
1	User	*"I have $30,000 USD, where should I invest it?"*
2	Mediator chatbot	*"Well, for how long could you keep the money invested?"*
3	User	*"Say, for 2 years."*
4	Mediator chatbot	*"All right, then, considering the amount and the time period, why don't you simulate this investment in a savings account?"*
5	User	*"Sure! I would love to."*
6	Mediator chatbot	*"Ok, I will invite the savings account to this group."*
7	Savings account chatbot	<< Savings account chatbot joins the group >>
8	Mediator chatbot	*"Hi Savings Account expert, could you please simulate the return of investment of $30,000 in 2 years?"*
9	Savings account chatbot	*"Sure, just a minute..."*
10	Savings account chatbot	*"Well, at the end, one would have $32,500 USD."*
11	Mediator chatbot	*"Thank you. Well, it seems a good idea given the economy right now."*
12	User	*"Thank you all."*
13	Savings account chatbot	*"You're welcome."*
14	Mediator chatbot	*"No problem. Let me know if I can help you with something else."*

The Table 10 shows an example that uses the mixed-initiative dialogue strategy, and a dialogue mediator to provide coordination control. In this example of an application, there are many types of intentions that should be answered: Q&A (question and answer) about definitions, investment options, and about the current finance indexes, simulation of investments, which is task-oriented and requires computation, and opinions, which can be highly subjective.

In Table 11, we present the interaction norms that were needed in Cognia. The `Trigger` column describes the event that triggers the `Behavior` specified in the third column. The `Pre-Conditions` column specifies what must happen in order to start the behavior execution. So, for instance, line 2, when the user sends an utterance in the chat group, an event is triggered **and**, if the utterance's topic is CDB (Certificate of Deposit which is a fixed rate investment) or if it is about the Savings Account investment option **and** the speech act is not `Query_Calculation` **and** the CDB and Savings Account members are not in the chat, **then** the behavior is activated. The bot members that implement these behaviors are called *cdbguru* and *poupancaguru*. Therefore these names are used when there is a mention.

Note that these interactions norms are not explicitly defined as obligations, permissions, and prohibitions. They are implict from the behavior described. During this implementation, we did not worry about explicitly defining the norms, because the goal was to evaluate the overall architecture, not to enhance the state of the art on norms specification for conversational systems. In addition, CognIA has only the presented interaction norms defined in Table 11, which is a very small set that that does not required model checking or verification of conflicts.

Table 11: Cognia Interaction Norms

Trigger	**Pre-Conditions**	**Behavior**
On group chat creation	Cognia chatbot is available	Cognia chatbot joins the chat with the *mediator* role and user joins the chat with the *owner_user* role
On utterance sent by user	Utterance's topic is CDB (*cdbguru*) **or** Savings Account (*poupancaguru*) **and** speech act is not *Query_Calculation* **and** they are not in the chat	Cognia invites experts to the chat and repeats the utterance to them
On utterance sent by user	Utterance's topic is CDB (*cdbguru*) **or** Savings Account (*poupancaguru*) **and** speech act is not *Query_Calculation* **and** they are in the chat	Cognia **waits for while** and *cdbguru* or *poupancaguru* respectively handles the utterance. If they don't understand, they **don't reply**
On utterance sent by the experts	If Cognia is waiting for them **and** has received both replies	Cognia does not wait anymore
On utterance sent	Utterance mentions *cdbguru* **or** *poupancaguru*	*cdbguru* or *poupancaguru* respectively handles the utterance
On utterance sent	Utterance mentions *cdbguru* **or** *poupancaguru* **and** they don't reply after a while **and** speech act is *Query_Calculation*	Cognia sends *I can only chat about investments...*
On utterance sent	Utterance mentions *cdbguru* **or** *poupancaguru* **and** they don't reply after while **and** speech act is not *Query_Calculation*	Cognia sends *I didn't understand*
On utterance sent	Utterance's speech act is *Query_Calculation* **and** period or initial amount of investment were **not** specified	Cognia asks the user the missing information
On utterance sent	Utterance's speech act is *Query_Calculation* **and** period and initial amount of investment were specified **and** the experts are **not** in the chat	Cognia invites experts to the chat and repeats the utterance to them

On utterance sent	Utterance's speech act is *Query_ Calculation* **and** period and initial amount of investment were specified **and** the experts are in the chat	Cognia repeats the utterance to experts
On utterance sent	Utterance's speech act is *Query_ Calculation*	Cognia extracts variables and saves the context
On utterance sent	Utterance's speech act is *Query_ Calculation* **and** the experts are in the chat **and** the experts are mentioned	Experts extract information, save in the context, compute calculation and send information
On utterance sent	Utterance's speech act is *Inform_ Calculation* **and** Cognia received all replies	Cognia compares the results and inform comparison
On utterance sent	Utterance mentions a chatbot but has **no** other text	The chatbot replies *How can I help you?*
On utterance sent	Utterance is not understood **and** speech act is *Question*	The chatbot replies *I don't know... I can only talk about topic X*
On utterance sent	Utterance is not understood **and** speech act is **not** *Question*	The chatbot replies *I didn't understand*
On utterance sent	Utterance's speech act is one of { *Greetings, Thank, Bye* }	All chatbots reply to utterance
On group chat end		All chatbots leave the chat, and the date and time of the end of chat is registered

5.2.1 Instantiating SABIA to develop CognIA

We instantiated SABIA to develop CognIA as follows: the `Mediator`, `Savings Account`, `CDB` and `User Actors` are the `Members` of the `Chat Group`. The `Hub` was implemented using two servers: `Socket.io` and `Node.JS` which is a socket client of the `Socket.io` server. The CognIA system has also one `Socket Client` for receiving the broadcast and forwarding to the `Group Chat Manager`. The former will actually do the broadcast to every member after enforcing the norms that applies specified in Table 11. Each `Member` will behave according to this table too. For each user of the chat group, on a mobile or a desktop, there is its corresponding actor represented by the `User Actor` in the figure. Its main job is to receive Akka's broadcast and forward to the Socket.io server, so it can be finally propagated to the users.

All the intents, actions, factual answers, context and logging data are saved in DashDB (a relational Database-as-a-Service system). When an answer is not retrieved, a service which executes the module `Search Finance on Social Media` on a separate server is called. This service was implemented with the assumption that finance experts post relevant questions and answers on social media. Further details are explained in the *Action execution* subsection.

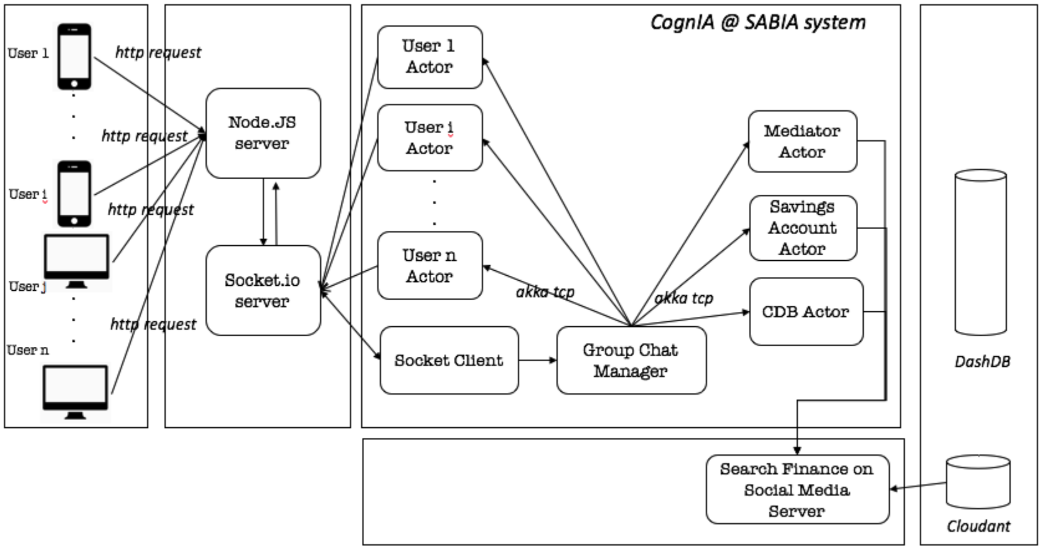

Figure 4: Cognia Deployment View

5.2.2 Topic Classifier

We built a small dictionary-based topic classifier to identify if an utterance refers to finance or not, and if it refers to the two investment options (CDB or Savings Account) or not.

5.2.3 Dependency Parsing

The dependency parsing is extremely important for computing the return of investment when the user sends an utterance with this intention. Our first implementation used regular expressions which led to a very fragile approach. Then we used a TensorFlow implementation[11] [48] of a SyntaxNet model for Portuguese and used it to generate the dependency parse trees of the utterances. The SyntaxNet model is a feed-forward neural network that operates on a task-specific transition system and achieves the state-of-the-art on part-of-speech tagging, dependency parsing and sentence compression results [49]. Below we present output[12] of the service for the utterance:

"I want to invest 10 thousands in 40 months":

```
{ "original": "I would like to invest 10 thousands in 40 months",
  "start_pos": [
    23,
    32],
  "end_pos": [
    27,
```

[11] TensorFlow is an open-source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. URL: https://www.tensorflow.org/

[12] Converted from Brazilian Portuguese to English.

```
    33],
  "digits": [
    10000,
    40],
  "converted": "I would like to invest 10000 in 40 months",
  "tree": {
    "like_VERB_ROOT": {
        "I_PRON_nsubj": {},
        "would_MD_aux":{
            "invest_VERB_xcomp":{
                "to_TO_aux": {},
                "10000_NUM_dobj": {},
                "in_IN_prep": {
                    "months_NOUN_pobj":{
                        "40_NUM_num": {}}}}}}}}
```

The service returns a `JSON Object` containing six fields: `original, start_pos, end_pos, digits, converted` and `tree`. The `original` field contains the original utterance sent to the service. The `converted` field contains the utterance replaced with decimal numbers, if the case (for instance, `"10 thousands"` was converted to `"10000"` and replaced in the utterance). The `start_pos` and `end_pos` are arrays that contain the start and end char positions of the numbers in the converted utterance. While the `tree` contains the dependency parse tree for the converted utterance.

5.2.4 Frame Parsing

Given the dependency tree, we implemented the frame parsing which first extracts the entities and features from the utterance and saves them in the context. Then, it replaces the extracted entities and features for reserved characters.

Algorithm 1 extract_period_of_investment (utteranceTree)

```
numbersNodes ← utteranceTree.getNumbersNodes();
foreach(numberNode in numbersNodes) do
    parentsOfNumbersNode ← numbersNode.getParents()
    foreach(parent in parentsOfNumbersNodes) do
        if ( parent.name contains { "day", "month", "year"} ) then
            parentOfParent ← parent.getParent()
            if ( parentOfParent is not null and
                parentOfParent.getPosTag==Verb and
                parentOfParent.name in investmentVerbsSet ) then
                return numberNode
```

Therefore an utterance like `"I would like to invest 10 thousands in 3 years"` becomes `"I would like to invest #v in #dt years"`. Or `"10 in 3 years"` becomes `"#v in #dt years"`, and both intents have the same intent class.

For doing that we implemented a few rules using a depth-first search algorithm combined with the rules as described in Algorithm 1, Algorithm 2 and Algorithm 3. Note that our parser works only for short texts on which the user's utterance mentions only one period of time and/ or initial amount of investment in the same utterance.

Algorithm 2 extract_initial_amount_of_investment (utteranceTree)

```
numbersNodes ← utteranceTree.getNumbersNodes();
foreach(numberNode in numbersNodes) do
    parentsOfNumbersNode ← numbersNode.getParents()
    foreach(parent in parentsOfNumbersNodes) do
        if ( parent.name does not contain { "day", "month", "year"} ) then
            return numberNode
```

Algorithm 3 frame_parsing(utterance, utteranceTree)

```
period ← extract_period_of_investment (utteranceTree)
save_period_of_investment(period)
value ← extract_initial_amount_of_investment (utteranceTree)
save_initial_amount_of_investment(value)
new_intent ← replace(new_intent, period, "#dt")
new_intent ← replace(new_intent, value, "#v")
```

5.2.5 Speech Act Classifier

In CognIA we have complemented the speech act classes with the ones related to the execution of specific actions. Therefore, if the chatbot needed to compute the return of investment, then, once it is computed, the speech act of the reply will be `Inform_Calculation` and the one that represents the query for that is `Query_Calculation`. In table 12 we list the specific ones.

Table 12: CognIA Specific Speech Acts Classes

Speech Act	**Example**
QUERY DEFINITION	*What is CD?*
INFORM DEFINITION	*A certificate of deposit (CD) is a time deposit, you lend money to the bank so it can use your money to invest somewhere else.*
QUERY NEWS	*Nowadays, is is better to invest in CDB or in the Savings Account?*
INFORM NEWS	*Depends, but nowadays CDs are good options.*
QUERY CALCULATION	*What if I invest 10 thousands for 3 years?*
INFORM CALCULATION	*At the end, you will have 12 thousands in the Savings Account and 12,5 thousand with a CD.*

5.2.6 Intention Classifier

Given that there is no public dataset available with financial intents in Portuguese, we have employed the incremental approach to create our own training set for the Intent Classifier. First, we applied the Wizard of Oz method and from this study, we have collected a set of 124 questions that the users asked. Next, after these questions have been manually classified into a set of intent classes, and used to train the first version of the system, this set has been increased both in terms of number of classes and samples per class, resulting in a training set with 37 classes of intents, and a total 415 samples, with samples per class ranging from

3 to 37.

We have defined our classification method based on features extracted from word vectors. Word vectors consist of a way to encode the semantic meaning of the words, based on their frequency of co-occurrence. To create domain-specific word vectors, a set of thousand documents are needed related to desired domain. Then each intent from the training set needs to be encoded with its corresponding mean word vector. The mean word vector is then used as feature vector for standard classifiers.

We have created domain-specific word vectors by considering a set 246,945 documents, corresponding to of 184,001 Twitter posts and and 62,949 news articles, all related to finance [13].

The set of tweets has been crawled from the feeds of blog users who are considered experts in the finance domain. The news article have been extracted from links included in these tweets. This set contained a total of 63,270,124 word occurrences, with a vocabulary of 97,616 distinct words. With the aforementioned word vectors, each intent from the training set has been encoded with its corresponding mean word vector. The mean word vector has been then used as feature vector for standard classifiers.

As the base classifier, we have pursued with a two-step approach. In the first step, the main goal was to make use of a classifier that could be easily retrained to include new classes and intents. For this reason, the first implementation of the system considered an 1-nearest-neighbor (1NN) classifier, which is simply a K-nearest-neighbor classifier with K set to 1. With 1NN, the developer of the system could simply add new intents and classes to the classifier, by means of inserting new lines into the database storing the training set. Once we have considered that the training set was stable enough for the system, we moved the focus to an approach that would be able to provide higher accuracy rates than 1NN. For this, we have employed Support Vector Machines (SVM) with a Gaussian kernel, the parameters of which are optimized by means of a grid search.

5.2.7 Action Classifier

We manually mapped the intent classes used to train the intent classifier to action classes and the dependent entities and features, when the case. Table 13 summarizes the number of intent classes per action class that we used in CognIA.

5.2.8 Action Execution

For the majority of action classes we used SABIA's default behavior. For instance, `Greet` and `Bye` actions classes are implemented using rapport, which means that if the user says *"Hi"* the chatbot will reply *"Hi"*.

The `Search News`, `Compute` and `Ask More` classes are the ones that require specific implemention for CognIA as following:

- Search News: search finance on social media service [50], [51] receives the utterance as input, searches on previously indexed Twitter data for finance for Portuguese and return to the one which has the highest score, if found.

[13] We have also evaluated word vectors created with all texts from the Wikipedia in Portuguese, but after conducting some preliminary experiments we observed that the use of tweets and news articles presented better results. We believe this happens due to (i) the more specific domain knowledge in the news articles and (ii) the more informal language used in tweets, which tend to be similar to texts sent to chatbots.

Table 13: Action Classes Mapping

Action Class	#Intent Classes	Entities	Features
Greet	1		
Thank	1		
Bye	1		
Get Definition	60		
Search News	45		
Compute	10	savings account, certificate of deposit	period of time, initial value
Ask More	79	savings account, certificate of deposit	period of time, initial value
Send Information	2		
Send Refuse	1		
No Action	1		

- Ask More: If the user sends an utterance that has the intention class of simulating the return of investment, while not all variables to compute the return of investment are extracted from the dialogue, the mediator keeps asking the user these information before it actually redirects the query to the experts. This action then checks the state of the context given the specified intent flow as described in (2) and (3) in section 4 to decide which variables are missing. For CognIA we manually added these dependencies on the database.

- Compute: Each expert Chatbot implements this action according to its expertise. The savings account chatbot computes the formula (4) and the certificate of deposit computes the formula (5). Both are currently formulas for estimating in Brazil.

$$RoI_{SA} = IV + IV * (R + TR) \tag{4}$$

  where RoI_{SA} is the return of investment for the savings account, IV is the initial value of investment, R is the savings account interest rate and TR is the savings account rate base[14].

$$RoI_{CD} = IV + (IV * ID * P^d) - IT \tag{5}$$

  where RoI_{CD} is the return of investment for certificate of deposit, IV is the initial value of investment, ID is the Interbank Deposit rate (DI in Portuguese), P is the ID's percentual payed by the bank (varies from 90% to 120%), d is the number of days the money is invested, and finally IT is the income tax on the earnings.

[14] The base rate is the interest rate that is paid each month regardless of whether your investment meets the conditions for the payment of the extra bonus rate.

Table 14: Action Execution

Action Class	**Task and Answer Generation**
Greet	rapport
Thank	send *you're welcome*
Bye	rapport
Get Definition	answer manually pre-defined or search finance on social media server
Search News	search finance on social media server
Compute	loads period of time and initial value from context to compute per chatbot expert
Ask More	search on the intent flow graph
Send Refuse	answer manually pre-defined
No Action	no answer

5.3 Intention Classifier Accuracy

In Table 15 we present the comparison of some distinct classification on the first version of the training set, i.e. the set used to deploy the first classifier into the system. Roughly speaking, the 1NN classifier has been able to achieve a level of accuracy that is higher than other well-known classifiers, such as Logistic Regression and Naïve Bayes, showing that 1NN is suitable as a development classifier. Nevertheless, a SVM can perform considerable better than 1NN, reaching accuracies of about 12 percentage points higher, which demonstrates that this type of base classifier is a better choice to be deployed once the system is stable enough. It is worth mentioning that these results consider the leave-one-out validation procedure, given the very low number of samples in some classes.

Table 15: Evaluation of different classifiers in the first version of the training set

	Precision	**Recall**	**F1**	**Accuracy**
1NN	0.85	0.84	0.84	0.84
Logistic Regression	0.66	0.68	0.64	0.68
Naïve Bayes	0.80	0.79	0.78	0.79
SVM	0.97	0.96	0.96	0.96

Table 16: Evaluation of different classifiers in the most recent version of the training set

	Precision	**Recall**	**F1**	**Accuracy**
1NN	0.81	0.80	0.80	0.80
Logistic Regression	0.75	0.77	0.75	0.77
Naïve Bayes	0.79	0.77	0.76	0.77
SVM	0.96	0.95	0.95	0.95

As we mentioned, the use of an 1NN classifier has allowed the developer of the system to easily add new intent classes and samples whenever they judged it necessary, so that the system could present new actions, or the understanding of the intents could be improved. As a consequence, the initial training set grew from 37 to 63 classes, and from 415 to 659 samples, with the number of samples per class varying from 2 to 63. For visualizing the impact on the accuracy of the system, in Table 16 we present the accuracy of the same classifiers used in the previous evaluation, in the new set. In this case, we observe some drop in accuracy for 1NN, showing that this classifier suffers in dealing with scalability. On the other hand, SVM has shown to scale very well to more classes and samples, since its accuracy kept at a very similar level than that with the other set, with a difference of about only 1 percentage point.

5.4 Testing SABIA

In this section, we describe the validation framework that we created for integration tests. For this, we developed it as a new component of SABIA's system architecture and it provides a high level language which is able to specify interaction scenarios that simulate users interacting with the deployed chatbots. The system testers provide a set of utterances and their corresponding expected responses, and the framework automatically simulates users interacting with the bots and collect metrics, such as time taken to answer an utterance and other resource consumption metrics (e.g., memory, CPU, network bandwidth). Our goal was to: (i) provide a tool for integration tests, (ii) to validate CognIA's implementation, and (iii) to support the system developers in understanding the behavior of the system and which aspects can be improved. Thus, whenever developers modify the system's source code, the modifications must first pass the automatic test before actual deployment.

5.4.1 Test framework

The test framework works as follows. The system testers provide a set $D = \{d_1, ..., d_n\}$ of dialogues as input. Each dialogue $d_i \in D$ is an ordered set whose elements are represented by $< u, R >$, where u is the user utterance and R is an ordered set of pairs $< r, c >$ that lists each response r each chatbot c should respond when the user says u. For instance, Table 17 shows a typical dialogue (d_1) between a user and the CognIA system. Note that we are omitting part of the expected answer with "..." just to better visualize the content of the table.

Table 17: Content of dialogue d_1 (example of dialogue in CognIA)

User utterance u	**rId**	**Expected response** r	**Chatbot** c
hello	1	Hello	Mediator
what is cdb?	2	@CDBExpert what is cdb?	Mediator
	3	CDB is a type of investment that...	CDB Expert
which is better: cdb or savings account?	4	I found a post in the social media for....	Mediator
i would like to invest R$ 50 in six months	5	@SavingsAccountExpert and @CDBExpert, could you do a simulation...	Mediator
	6	If you invest in Savings Account, ...	Savings Account Exp.
	7	If you invest in CDB,...	CDB Expert
	8	Thanks	Mediator
	9	@User, there is no significant difference..	Mediator

so i want to invest R$ 10000 in 2 years	10	@SavingsAccountExpert and @CDBExpert, could you do a simulation...	Mediator
	11	If you invest in Savings Account,...	Savings Account Exp.
	12	If you invest in CDB,...	CDB Expert
	13	Thanks	Mediator
	14	@User, in that case, it is better...	Mediator
what if i invest R$10,000 in 5 years?	15	@SavingsAccountExpert and @CDBExpert, could you do a simulation...	Mediator
	16	If you invest in Saving Account,...	Savings Account Exp.
	17	If you invest in CDB,...	CDB Expert
	18	Thanks	Mediator
	19	@User, in that case, it is better...	Mediator
how about 15 years?	20	@SavingsAccountExpert and @CDBExpert, could you do a simulation...	Mediator
	21	If you invest in Savings Account,...	Savings Account Exp
	22	If you invest in CDB,...	CDB Expert
	23	Thanks	Mediator
	24	@User, in that case, it is better...	Mediator
and 50,0000?	25	@SavingsAccountExpert and @CDBExpert, could you do a simulation...	Mediator
	26	If you invest in Savings Account,...	Savings Account Exp.
	27	If you invest in CDB,...	CDB Expert
	28	Thanks	Mediator
	29	@User, in that case, it is better..	Mediator
I want to invest in 50,000 for 15 years in CDB	30	Sure, follow this link to your bank...	Mediator
thanks	31	You are welcome.	Mediator

The testers may also inform the number of simulated users that will concurrently use the platform. Then, for each simulated user, the test framework iterates over the dialogues in D and iterates over the elements in each dialogue to check if each utterance u was correctly responded with r by the chatbot c. There is a maximum time to wait. If a bot does not respond with the expected response in the maximum time (defined by the system developers), an error is raised and the test is stopped to inform the developers about the error. Otherwise, for each correct bot response, the test framework collects the time taken to respond that specific utterance by the bot for that specific user and continues for the next user utterance. Other consumption resource metrics (memory, CPU, network, disk). The framework is divided into two parts. One part is responsible to gather resource consumption metrics and it resides inside SABIA. The other part works as clients (users) interacting with the server. It collects information about time taken to answer utterances and checks if the utterances are answered correctly.

By doing this, we not only provide a sanity test for the domain application (CognIA) developed in SABIA framework, but also a performance analysis of the platform. That is, we can: validate if the bots are answering correctly given a pre-defined set of known dialogues, check if they are answering in a reasonable time, and verify the amount of computing resources that were consumed to answer a specific utterance. Given the complexity of CognIA, these tests enable debugging of specific features like: understanding the amount of network bandwidth to use external services, or analyzing CPU and memory consumption when responding a specific utterance. The later may happen when the system is performing more complex calculations to indicate the investment return, for instance.

5.4.2 Test setup

CognIA was deployed on IBM Bluemix, a platform as a service, on a Liberty for Java Cloud Foundry app with 3 GB RAM memory and 1 GB disk. Each of the modules shown in Figure 4 are deployed on separate Bluemix servers. `Node.JS` and `Socket.IO` servers are both deployed as Node Cloud Foundry apps, with 256 MB RAM memory and 512 MB disk each. `Search Finance on Social Media` is on a Go build pack Cloud Foundry app with 128 MB RAM memory and 128 GB disk. For the framework part that simulates clients, we instantiated a virtual machine with 8 cores on IBM's SoftLayer that is able to communicate with Bluemix. Then, the system testers built two dialogues, i.e., $D = \{d_1, d_2\}$. The example shown in Table 17 is the dialogue test d_1. For the dialogue d_2, although it also has 10 utterances, the testers varied some of them to check if other utterances in the finance domain (different from the ones in dialogue d_1) are being responded as expected by the bots. Then, two tests are performed and the results are analyzed next. All tests were repeated until the standard deviation of the values was less than 1%. The results presented next are the average of these values within the 1% margin.

5.4.3 Results

Test 1: The first test consists of running both dialogues d_1 and d_2 for only one user for sanity check. We set `30 seconds` as the maximum time a simulated user should wait for a bot correct response before raising an error. The result is that all chatbots (`Mediator, CDBExpert, and SavingsAccountExpert`) responded all expected responses before the maximum time. Additionally, the framework collected how long each chatbot took to respond an expected answer.

In Figure 5, we show the results for those time measurements for dialogue d_1, as for the dialogue d_2 the results are approximately the same. The x-axis (`Response Identifier`) corresponds to the second column (`Resp. Id`) in Table 17. We can see, for example, that when the bot `CDBExpert` responds with the message `3` to the user utterance `"what is cdb?"`, it is the only bot that takes time different than zero to answer, which is the expected behavior. We can also see that the Mediator bot is the one that takes the longest, as it is responsible to coordinate the other bots and the entire dialogue with the user. Moreover, when the expert bots (CDBExpert and SavingsAccountExpert) are called by the Mediator to respond to the simulation calculations (this happens in responses 6, 7, 11, 12, 16, 17, 21, 22, 26, 27), they take approximately the same to respond. Finally, we see that when the concluding responses to the simulation calculations are given by the Mediator (this happens in responses 9, 14, 19, 24, 29), the response times reaches the greatest values, being 20 seconds the greatest value in response 19. These results support the system developers to understand the behavior of the system when simulated users interact with it and then focus on specific messages that are taking longer.

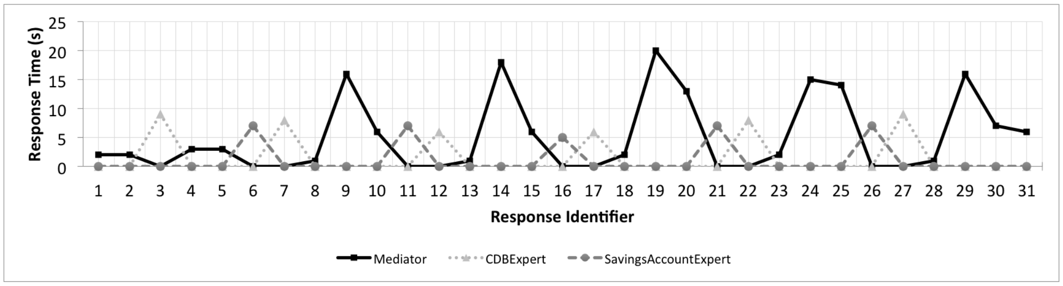

Figure 5: Single simulated user results for dialogue d_1.

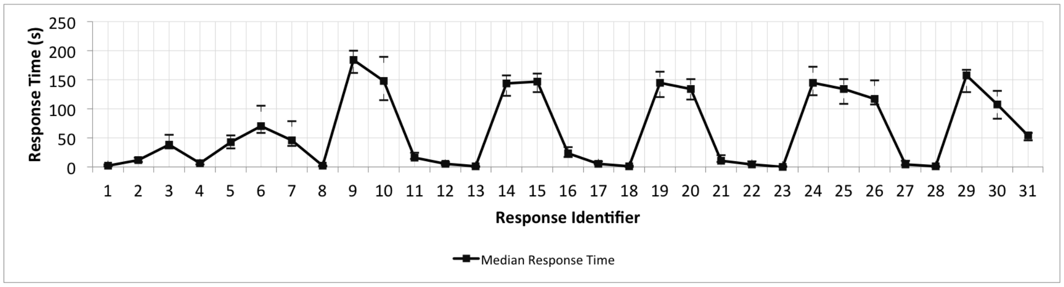

Figure 6: Eight concurrent simulated users.

Test 2: This test consists of running dialogue d_1, but now using eight concurrent simulated users. We set the maximum time to wait to 240 seconds, i.e., eight times the maximum set up for the single user in Test 1. The results are illustrated in Figure 6, where we show the median time for the eight users. The maximum and minimum values are also presented with horizontal markers. Note that differently than what has been shown in Figure 5, where each series represents one specific chatbot, in Figure 6, the series represents the median response time for the responses in the order (x-axis) they are responded, regardless the chatbot.

Comparing the results in Figure 6 with the ones in Figure 5, we can see that the bots take longer to respond when eight users are concurrently using the platform than when a single user uses it, as expected. For example, CDBExpert takes approximately 5 times longer to respond response 3 to eight users than to respond to one user. On average, the concluding responses to the simulation questions (i.e., responses 9, 14, 19, 24, 29) take approximately 7.3 times more to be responded with eight users than with one user, being the response 9 the one that presented greatest difference (11.4 times longer with eight users than with one). These results help the system developers to diagnose the scalability of the system architecture and to plan sizing and improvements.

6 Conclusions and Future Work

In this article, we explored the challenges of engineering MPCS and we have presented a hybrid conceptual architecture and its implementation with a finance advisory system.

We are currently evolving this architecture to be able to support decoupled interaction norms specification, and we are also developing a multi-party governance service that uses that specification to enforce exchange of compliant utterances.

In addition, we are exploring a micro-service implementation of SABIA in order to increase its scalability and performance, so thousands of members can join the system within thousands of conversations.

Acknowledgments

The authors would like to thank Maximilien de Bayser, Ana Paula Appel, Flavio Figueiredo and Marisa Vasconcellos, who contributed with discussions during SABIA and CognIA's implementation.

References

[1] Cicero. *On Duties.* 42BC.

[2] C. Danescu-Niculescu-Mizil, R. West, D. Jurafsky, J. Leskovec, and C. Potts. No country for old members: User lifecycle and linguistic change in online communities. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW'13, 2013.

[3] C. Jenks. *Social Interaction in Second Language Chat Rooms.* Edinburgh University Press, Edinburgh, 2014.

[4] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[5] Loebner prize, 1990. http://www.loebner.net/Prizef/loebner-prize.html.

[6] Markus M. Berg. *Modelling of Natural Dialogues in the Context of Speech-based Information and Control Systems.* PhD thesis, Universität zu Kiel, Kiel, Germany, July 2014.

[7] Facebook posts strong profit and revenue growth, 2016. https://www.wsj.com/articles/facebook-posts-strong-profit-and-revenue-growth-1469650289.

[8] T. Parsons. *The Structure of Social Action.* McGraw-Hill, New York, 1937.

[9] Wendy G. Ju. *The Design of Implicit Interactions.* PhD thesis, Stanford, CA, USA, 2008. AAI3313595.

[10] L. Suchman. *Human-machine reconfigurations: Plans and situated actions.* Cambridge University Press, Cambridge, 2007.

[11] V. Zue. Proc. of ieee. 88(8):1166–1180, 2000.

[12] J. Weizenbaum. *Computer Power and Human Reason: From Judgment to Calculation.* W.H. Freeman and Company, New York, NY, 1976. ISBN 0-7167-0464-1.

[13] R. Wallace. Alice. *Thompson*, 2002. p. 2.

[14] R. Wallace. The anatomy of alice, 2006. Parsing the Turing Test.

[15] Jacob Aron. Software tricks people into thinking it is human., 2016.

[16] H. Jenkins, S. Ford, and J. Green. *Spreadable media: Creating value and meaning in a networked culture.* NYU press, 2013.

[17] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. Introduction to ?this is watson? *IBM Journal of Research and Development*, 56, 2012.

[18] P. Olson. Get ready for the chat bot revolution: They?re simple, cheap and about to be everywhere., August 2016.

[19] Harvey Sacks, Emanuel A. Schegloff, and Gail Jefferson. A simplest systematics for the organization of turn-taking for conversation. *Language*, 50(4):696–735, 1974.

[20] D. Bohus and E. Horvitz. Decisions about turns in multiparty conversation: From perception to action. In *Proceedings of the 13th International Conference on Multimodal Interfaces*, ICMIÍ1, pages 153–160, New York, NY, USA, 2011. ACM.

[21] Nicholas R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44:35–41, 2001.

[22] N. H. Minsky and D. Rozenshtein. A law-based approach to object- oriented programming. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA'87, pages 482–493, New York, NY, 1987. ACM Press.

[23] N.H. Minsky and V. Ungureanu. Law-governed interaction: a coordination and control mechanism for heterogeneous distributed systems. *ACM Transactions on Software Engineering Methodologies*, 9:273–305, 2000.

[24] T. Murata and N.H. Minsky. On monitoring and steering in large-scale multi-agent systems. In *The 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems*, SELMAS'03, 2003.

[25] M. Esteva. *Electronic Institutions: from specification to development.* PhD thesis, Institut d'Investigaci en Intelligncia Artificial, Catalonia, Spain, 2003.

[26] R. Paes, G. R. Carvalho, C.J.P. Lucena, P. S. C. Alencar, Almeida H.O., and V. T. Silva. Specifying laws in open multi-agent systems. In *ANIREM*, AAMAS'05, 2005.

[27] R. B. Paes, G. R. Carvalho, M. A. C. Gatti, C.J.P. de Lucena, J.-P. BRIOT, and R. Choren. *Enhancing the Environment with a Law-Governed Service for Monitoring and Enforcing Behavior in Open Multi-Agent Systems*, volume 4389. Springer-Verlag, Berlim, 2007.

[28] Hadeel Al-Zubaide and Ayman A. Issa. Ontbot: Ontology based chatbot. In *Fourth International Symposium on Innovation in Information and Communication Technology (ISIICT'11)*, Jordan, 2011.

[29] A. Bordes and J. Weston. Learning end-to-end goal-oriented dialog. *CoRR*, abs/1605.07683, 2016.

[30] H. Wright. Automatic utterance type detection using suprasegmental features. In *ICSLP'98*, volume 4, page 1403, Sydney, Australia, 1998.

[31] N. Roy, J. Pineau, and S. Thrun. Spoken dialogue management using probabilistic reasoning. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, ACL '00, pages 93–100, Stroudsburg, PA, USA, 2000. Association for Computational Linguistics.

[32] Peter Lee. Learning from tay's introduction, 2016.

[33] Chunxi Liu, Puyang Xu, and Ruhi Sarikaya. Deep contextual language understanding in spoken dialogue systems. ISCA - International Speech Communication Association, September 2015.

[34] Iulian Vlad Serban, Ryan Lowe, Peter Henderson, Laurent Charlin, and Joelle Pineau. A survey of available corpora for building data-driven dialogue systems. *CoRR*, abs/1512.05742, 2015.

[35] Iulian Vlad Serban, Alessandro Sordoni, Yoshua Bengio, Aaron C. Courville, and Joelle Pineau. Hierarchical neural network generative models for movie dialogues. *CoRR*, abs/1507.04808, 2015.

[36] Alessandro Sordoni, Michel Galley, Michael Auli, Chris Brockett, Yangfeng Ji, Margaret Mitchell, Jian-Yun Nie, Jianfeng Gao, and Bill Dolan. A neural network approach to context-sensitive generation of conversational responses. *CoRR*, abs/1506.06714, 2015.

[37] S. Sukhbaatar, A Szlam, J. Weston, and R. Fergus. Weakly supervised memory networks. *CoRR*, abs/1503.08895, 2015.

[38] Hamid R. Chinaei, Brahim Chaib-draa, and Luc Lamontagne. *Application of Hidden Topic Markov Models on Spoken Dialogue Systems*, pages 151–163. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[39] R. Higashinaka. Towards taxonomy of errors in chat-oriented dialogue systems. In *Proceedings of the SIGDIAL 2015 Conference*, Association for Computational Linguistics, pages 87–95, Prague, Czech Republic, 2015.

[40] Caleb Meier, Puja Valiyil, Aaron Mihalik, and Adina Crainiceanu. Rya optimizations to support real time graph queries on accumulo, 2015.

[41] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. In *In International Conference on Learning Representations (ICLR'2015)*, 2015.

[42] Slav Petrov, Dipanjan Das, and Ryan McDonald. A universal part-of-speech tagset. In *IN arXiv:1104.2086*, 2011.

[43] J. Nivre. Algorithms for deterministic incremental dependency parsing. *Comput. Linguist.*, 34(4):513–553, December 2008.

[44] FIPA. The foundation for intelligent physical agents, 2002.

[45] J. Armstrong. Erlang. *Commun. ACM*, 53(9):68–75, September 2010.

[46] C. Hewitt, P. Bishop, and R. Steiger. A universal modular actor formalism for artificial intelligence. In *Proceedings of the 3rd International Joint Conference on Artificial Intelligence*, IJCAI'73, pages 235–245, San Francisco, CA, USA, 1973. Morgan Kaufmann Publishers Inc.

[47] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns.* Addison-Wesley, 1994. Template Method: pages 325–330.

[48] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[49] D. Andor, C. Alberti, D. Weiss, A. Severyn, A. Presta, K. Ganchev, S. Petrov, and M. Collins. Globally normalized transition-based neural networks. *CoRR*, abs/1603.06042, 2016.

[50] F. Figueiredo, M. A. Vasconcelos, and C. Pinhanez. Does your link relate to my question? Understanding question answering through links on social media. In *Technical Report*, 2016.

[51] P. Cavalin, F. Figueiredo, M. Gatti de Bayser, L. Moyano, H. Candello, A. Appel and R. Souza Building a question-answering corpus using social media and news articles. In *International Conference on the Computational Processing of the Portuguese Language*, PROPOR, Tomar, Portugal, 2016.