Ravel: A MAS orchestration platform for Human-Chatbots Conversations

Maira Gatti de Bayser, Claudio Pinhanez, Heloisa Candello, Marisa Affonso Vasconcelos, Mauro Pichiliani, Melina Alberio Guerra, Paulo Cavalin, and Renan Souza

IBM Research, Brazil

Abstract. This paper presents *Ravel*, a multiagent systems (MAS) platform aimed to integrate natural language understanding components with orchestration components of dialogues between human beings and agents. *Ravel* enables the specification of (social) conversations norms, using deontic logic, for use in contexts where multiple agents and human users are conversing in natural language. We demonstrate the usefulness and versatility of *Ravel* in a chat-based finance adviser system designed as a chat group of five participants: four collaborative chatbots with two different roles (mediator and expert) and a human or chatbot user.

Keywords: Multi-Agent System · Norms · Social Chatbots · Dialogue

1 Introduction

In the last two decades, the Agent-Oriented Software Engineering community has created several processes, methods, and tools to develop agent-based systems [6] [10] [17] [2], which has been successfully applied to many applications and domains. More recently, the Natural Language Processing community has achieved excellent results on machine learning, using SVM and Neural Networks, for natural language understanding at several tasks, from intent detection to turn-taking in dialogue systems [22] [12] [7] [3].

There is still though a lack of work which combines both solutions into a common platform and enables developers to build agents which not only understand natural language but also interact with multiple humans and other agent simultaneously in conversation contexts such as chats (aka chatbots). Most of current industry chatbots, like *Siri* and *Cortana*, for instance, are dyadic chatbots, that is, they are designed to handle only one speaker at a time. A key challenge faced by chatbots in multi-party conversations is *turn-taking*[5] [4], that is, determining whether they should or not produce an utterance at any moment of the dialogue.

Therefore, the main contribution of our work is a MAS orchestration platform for multi-party chat-oriented dialogues, called *Ravel*. Ravel was evaluated with an application called *finch*, an investment adviser system with four chatbots which cooperate to help users to explore low-risk investment options. To orchestrate *finch* conversations, around 15 conversation norms based on deontic modes [23][21] are successfully applied by Ravel even in the context of several simultaneous users with an average chatbot response time of less than 1 sec.

2 *finch*: Finance Cooperative Chatbots

In this section we present *finch*: an interactive investment adviser system, which helps users to make more informed low-risk investment decisions in Brazil. It was designed as a *WhatsApp*-like chat where a user can converse with four chatbots representing three investment products playing the role of experts, and one to mediate the conversation, playing the role of a friendly finance investment counselor. The three chatbot experts are gurus on low-risk Brazilian financial products: *SA*, expert in savings account; *TB*, expert in treasury bonds; and *CD*, expert in certificate of deposit. The mediator chatbot is the *In* (investment expert) which moderates the conversation, helps to summarize it, and prompts the user to make more questions.

The conversation start with the investment bot (In) informing the user the possibility to ask about concepts or to ask for simulation on return of investments for each investment option. The user then asks to simulate for four thousands as the initial amount in three years. The idea is that after the first utterance is sent by the investment chatbot (In), the bots expect the user reply and the bots must not reply to the investment bot . In a chat group, since all the utterances are broadcasted to all participants, the investment chatbot will receive its own utterance back, so it must not reply to itself.

Moreover, when the user replies to some utterances, we have a situation where the user is (still) allowed to reply, the investment chatbot (In) is ought to reply and the investment option experts (TB, SA and CD) are forbidden to reply. We expect the investment chatbot (In) to mediate the simulation by asking any partial information needed before the experts bots can compute the simulation and reply. So, for instance, if the user sends a message such as *"I would like to invest in 3 years"* without mentioning the investment amount, then the investment chatbot (In) has to ask the user that amount. Following, investment chatbot (In) then produces a request to all the expert chatbots to simulate the investment, blocking the user, and giving them permission to speak. When one of the expert chatbots replies, they are no longer expected to interact in the chat unless a new utterance requires. The same rule is applied for each expert chatbot which answers afterwards. When finally all experts have answered, the investment chatbot (In) then has an obligation to inform which investment option is the best according to the simulation results.

Another interesting example of the complexity of turn-taking which has to be handled is when the user asks questions about some of the investment products and related concepts. Given a question about one of the investments from the user, the expert chatbot associated with that concept must answer. If it is not possible to determine the specific topic, the chatbot which answered the last question keeps with the obligation to answer. If the user mention the name of a specific chatbot (using the "@name" moniker), that chatbot has an obligation to reply too.

For implementing *finch* with the said requirements, there is a need to engineer a MAS-based platform with an environment where each chatbot is implemented as an agent and requires sets of natural language understanding components in order to interact in a chat group. In addition, it is needed that the interaction is coherent and that the chatbots do not reply to each other, unless explicitly required by another chatbot.

3 Background on MAS and Related Work

In this section we present the state of the art on governance of multi-agent systems, since it is related to the governance of the conversation of chatbots (considering each chatbot as an agent). Given a law as a set of rules which governs the interaction, Minsky et al. proposed the *Law-Governed Interaction (LGI)* conceptual model about the role of interaction laws on distributed systems, and conducted further work and experimentation based on those ideas [15]. However, his approach lacked the ability to express high level information of social systems. Following the same approach but more at the agent social level, the *Electronic Institution* framework [1] provided support for interaction rules. The Electronic Institution has a set of high-level abstractions which allows for the specification of laws using concepts such as agent roles, norms, and scenes. Also at the agent social level, in open multi-agent systems, the *XMLaw* description language and the *M-Law* framework [19] [18] were proposed and developed to support law-governed mechanisms. More recent work regarding coordination in open multi-agent systems can be found here [2]. To our knowledge, none of them support the kind of hybrid human-agent issues with natural language discussed in this paper.

At the middleware and platforms level for multi-agent systems, the most known works are *JADE* [6], *Jadex* [20], *Jack* [9] and *Jason* [8]. Those platforms let the agent developer focus only at the logic level of the agent, not having to worry about the communication management nor having to consider the agent address in the network. Most of them are *FIPA-ACL*-compliant with regard to the message communication protocol between agents, to ensure interoperability in an open environment. In our case the high-level communication is natural language and the platform must be able to receive as an input a text in natural language.

Finally with regard to chatbot engines, or platforms to support the development of conversational systems[1], there is a lack of research directed to building inter-message coherence rules integrated with natural language.

[1] For instance, IBM Watson Conversation Service: https://www.ibm.com/watson /services/conversation/

4 The *Ravel* Platform

To address turn-taking and other conversation orchestration issues in multi-party hybrid chat-based systems, we have developed *Ravel*. It was designed as a MAS-based Microservices-driven architecture[16] with Natural Language Understanding (NLU) and Dialog components to enable agents to communicate with the users through dialogues with natural language. The MAS environment is composed of an agent which is a Communication Hub (CH) that enables the message exchange, the chatbots which are agents, a Connector, which connects the agents to the CH, and a Conversation Governance (CG) service to orchestrate the agents dialogues.

The NLU and Dialog components and the CG service are stateless microservices. Since the CG service is needed in the MAS environment and is one of the contributions of this paper we briefly detail it in this section. For a given input with the message and the current status of the conversation, it checks the state of the conversation according to the rules and it tags the message with three attributes: `isAllowed`, `isDenied` or `isRequired`. Each tag corresponds to the three modes of *deontic logic* [23] [13] [14][21]: the permission, prohibition or obligation norms, respectively.

For more information about the CG service, see [4]. It is implemented as an interpreter of a Domain Specific Language for Conversation Rules, (DSL-CR) which enables modeling, specification, and execution of multi-party conversation rules. In the DSL-CR grammar, a conversation is composed of at least three concepts: `Participant`, `Role` and `Protocol`. The `Participant` can be a chatbot or a person, while the `Role` defines the role of the participant in the conversation. The `Norm` is the deontic mode and can be a permission, obligation, or prohibition as discussed before. Whenever a `Message` is exchanged in the conversation, the norms which are currently active in the conversation are checked and, if the message is allowed according to those norms, it can (or must in case of obligations) be uttered by the participant (human or agent). This verification is made using the role of the participant which is specified in the utterance. The message features which can be used to create fine-grained rules depends on the communication model and may change according to the application domain. The `Message` follows the *FIPA-ACL* specification [11] for a subset of fields.

The `Turn` in the conversation is the exchange by one single participant of one message which contains one or more utterances. It represents an event which can change the state of the conversation or the set of norms which are active in the conversation such as an utterance arrival. The norms are defined in terms of for a given turn in a conversations as:

*An **obligation** requires the participant to pro-actively or reactively emit an utterance*;

*A **permission** allows the participant to pro-actively or reactively emit an utterance*;

*A **prohibition** forbids the participant to emit utterances, or states that they are not expected in that turn.*

For example, a permission like the following can be created:

Permission: *Whenever an utterance with the **inform** or **query** speech act and the **savings** topic is present, a participant with the **savings expert** role has the **permission** to reply.*

In addition, for each message that arrives, the GC service uses variables as *$sender*, *$last_sender* and *$receivers*, besides the participant roles, to identify the members that can be eligible to receive the activated norms (see please Appendix B[2] with examples of specification for *finch* application).

Both the agents and client applications connect to CH. Therefore, there are connectors which manage these connections in a way that the agent does not need to be aware of the existence of CH. In this way, CH intermediates the conversation so even traditional and legacy dyadic chatbots can be inserted into a multi-party context. The CH is the agent responsible for broadcasting the exchanged messages between the chatbots and the users and is a stateful microservice because it needs to manage the message queue (inbox): every time a message arrives, the CH uses its conversation identification to look for the participants of the given conversation.

The MAS environment is therefore event and message-oriented. The CH listens to some events in order to handle the incoming messages from chatbots and users. During inbox checking, a message is peeked and verified by CG Service. It decides, according to the defined rules, if the message is allowed to be broadcasted. If not allowed, that was because of a prohibition. If allowed, it is either a permission or obligation, and all agents connected to CH and listening to CH events receive the broadcasted messages, generate the responses and emit it back to the CH. For more details about CH behavior, please see Algorithm 1 and 2 in Appendix A.

5 *Ravel* Instantiation

We implemented *RESTful* agents using *Nodejs* and deployed them together with Ravel CH, connectors (one for each agent), and the Governance Service in a *Docker container*[3] in *IBM Cloud*[4]. The agents call the stateless microservices whenever they need any natural language understanding task. All the specific actions of the agents for *finch*, like the one which computes the return of an investment, are implemented within the agent with the corresponding role. The MAS environment relies on top of Socket.io[5] middleware. The CH is a socket server, while the connectors and client application are socket clients.

The norms and conversation protocols governing the conversation in *finch* are specified using 9 different roles which the user and the agents assume in conversations: person, bot, user, mediator, investment product expert, investment counselor, savings account adviser, certificate of deposit adviser, and treasury bonds adviser. The agents can play the roles of bot, mediator, expert, investment

[2] http://emas2018.dibris.unige.it/images/papers/EMAS18-19-appendix.pdf
[3] https://www.docker.com/what-docker
[4] http://www.ibm.com/bluemix
[5] Socket.io: https://socket.io/

counselor, savings account adviser, certificate of deposit adviser, and treasury bonds adviser. While the user plays the roles of person and user.

Regarding speech acts, the agents which played the role of experts were designed to answer to *greetings*, *thank*, *query*, *request*, *inform*, *agree*, *not understood*, and *bye*. Among the *query* speech act sub-classes, the agents could also perform simulation to compute the return of investment for a set of values given by the user using natural language. We specified the speech act and topic classes based on the classes recognized by the microservices which we trained for *finch*. There were in total 13 different speech acts. The topic classifier handles 6 different topics: investment in general, savings account, certificate of deposit, treasury bonds, all, or undefined based on a dictionary.

In *finch*, 15 norms were defined of which 6 were prohibitions, 2 were permissions, and 7 were obligations. These norms are used in 25 transitions, either for theirs activation or deactivation. For the conversation protocol, we designed the finite-state- automaton to contain only 2 states: one in which no norm is active and one in which at least one norm is active. The initial state of *finch* is one with active norms because the conversation starts with investment chatbot (In) having the obligation to talk. A subset of *finch* conversation rules specification is present in Appendix B for illustration.

The GC service is implemented as the following. Once a message arrives at GC Service, the best possible message descriptor is filtered by the message descriptor attributes and, in case a message matches to two or more descriptors, the one with the larger number of filters is selected. If still remains two or more descriptors, the first one is selected. The chosen message descriptor will trigger one of the specified transitions, which will activate the norms to be applied to the participants according to their roles, and also based on the variables regarding to sender, last sender and receivers message fields.

To illustrate, please see Appendix B. When the user requests the return on investment calculation, the *m3* descriptor is detected and triggers the transition *t1*, which activates the norms *oblig_to_mediator*, *proh_to_experts* and *perm_to_sender*. So, the *In* bot is required to reply and sends a message requesting the simulation to the investment experts. This new message is identified with the message descriptor *m4*, which triggers the transition *t2*, and now this transition deactivates some norms and activates the norms *oblig_to_receiver* that applies to participantes stored in *$receiver* variable, *perm_to_sender* that applies to *$sender* , *proh _to_experts_except_to_infor- m_calc* and *proh_to_mediator_except_to_finish_calc*.

Then, as expected, the three required experts replies, and transition *t3* is fired which allows the mediator to reply with a message suggesting the best option between the three investments, concluding the calculation results.

6 Experimental Results

We run a user study without human supervision in which 37 participants interacted with *finch*. 3717 messages were generated by the chatbots but only 963 were allowed and actually posted back in the chat. The average number of messages per dialogue was 100 (including the prohibited and the ones not displayed to the user) with a standard deviation of 49.49 and coefficient of variation of 0.49. The average number of allowed messages per dialogue was 26 with standard deviation of 12.77 and coefficient of variation of 0.49. The average chatbot response time per user message was 0.63 seconds with 0.4 as the standard deviation. There were on average 15 generated messages per minute with a 9.27 standard deviation, of each 4 on average were allowed per minute, with 2.67 standard deviation. From the 25 transitions specified, only 18 were actually activated in this experiment. Transition *t2* was the most frequent (18.88%), while transition *t3* happened 14.21% of the time. Within the 7 transitions which were not activated, 4 were transitions from or to the state with no active norms. This means that there was at least one active norm during all dialogues. The other 3 were transitions activated by a message with a name mention to a prohibited receiver, an inform from the mediator, and off-topic messages sent by the user. Those three types of messages did not happen. We performed qualitative analysis on the 37 dialogues with regard to inter-message coherence. Only 2 conversations demonstrated some level of incoherence. This happened because the user asked to simulate the return of investment for only one particular investment option. The results indicated that the initial set of norms we modeled were sufficient to enable inter-message coherence in 97% of the dialogues.

7 Conclusions and Future Work

We have presented in this paper *Ravel*, a MAS-based platform able to handle, in real scenarios, many of the key turn-taking issues in multi-party chat-oriented conversations. We have described how we successfully instantiated it to a real-world application with chatbots as finance advisors. There are, however, some challenges which are still open. For instance, given a set of rules which are specific to a turn, there is the problem to create generic rules which can be applied not only to that specific turn but to similar turns. There is a need of creating an automated method for this task. Hence, conversation rules learning from dialogue corpus is our future work.

References

1. Arcos, J.L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J., Sierra, C.: Engineering open environments with electronic institutions. Eng. Appl. Artif. Intell. **18**(2), 191–204 (Mar 2005)
2. Artikis, A., Sergot, M., Pitt, J., Busquets, D., Riveret, R.: Specifying and Executing Open Multi-agent Systems, pp. 197–212. Springer Int. Publishing (2016)
3. Asri, L., Schulz, H., Sharma, S., Zumer, J., Harris, J., Fine, E., Mehrotra, R., Suleman, K.: Frames: a corpus for adding memory to goal-oriented dialogue systems. In: Proc. of the 18th Annual SIGdial Meeting on Discourse and Dialogue. pp. 207–219. ACL, Saarbraecken, Germany (August 2017), http://aclweb.org/anthology/W17-5526

4. Gatti de Bayser, M., Alberio Guerra, M., Cavalin, P., Pinhanez, C.: Specifying and implementing multi-party conversation rules with finite-state-automata. In: Proc. of the AAAI Workshop On Reasoning and Learning for Human-Machine Dialogues, DeepDial'18 (Feb 2018)
5. Gatti de Bayser, M., Cavalin, P., Souza, R., Braz, A., Candello, H., Pinhanez, C., Briot, J.P.: A hybrid architecture for multi-party conversational systems. (2017)
6. Bellifemine, F., Poggi, A., Rimassa, G.: Jade: a fipa2000 compliant agent development environment. In: In: Proc. of Agents Fifth Int. Conf. on Autonomous Agents. pp. 216–217 (2001)
7. Bordes, A., Weston, J.: Learning end-to-end goal-oriented dialog. CoRR **abs/1605.07683** (2016)
8. Bordini, R., Hbner, J., Wooldridge, M.: Programming Multi-Agent Systems in AgentSpeak using Jason. Wiley Series in Agent Technology, Wiley (2007)
9. Busetta, P., Rnnquist, R., Hodgson, A., Lucas, A.: JACK Intelligent Agents - Components for Intelligent Agents in Java. AgentLink News (1999)
10. Gleizes, M.P., Gomez-Sanz, J. (eds.): AOSE X. Springer, Heidelberg (2010)
11. for Intelligent Physical Agents, F.F.: Fipa acl message structure specification (2002), http://www.fipa.org/specs/fipa00061/index.html
12. Liu, C., Xu, P., Sarikaya, R.: Deep contextual language understanding in spoken dialogue systems. In: INTERSPEECH 2015, 16th Annual Conf. of the Int. Speech Communication Association, Dresden, Germany, September 6-10, 2015. pp. 120–124 (2015)
13. Meyer, J.J.: Applications of Deontic Logic in Computer Science: A Concise Overview, pp. 17–40. John Wiley & Sons (1993)
14. Meyer, J.J., Wieringa, R. (eds.): Deontic Logic in Computer Science: Normative System Specification. John Wiley and Sons Ltd., Chichester, UK (1993)
15. Murata, T., Minsky, N.H.: On monitoring and steering in large-scale multi-agent systems. In: In: Selmas03́ 2nd Int. Workshop on Soft. Eng. for Large-Scale Multi-Agent Systems, AAMAS (2003)
16. Newman, S.: Building Microservices. O'Reilly Media (2015)
17. Padgham, L., Winikoff, M.: Developing Intelligent Agent Systems: A Practical Guide. John Wiley & Sons, Hoboken, NJ (2005)
18. Paes, R., Carvalho, G., Gatti, M., Lucena, C., Briot, J.P., Choren, R.: Enhancing the environment with a law-governed service for monitoring and enforcing behavior in open multi-agent systems, Lecture Notes in Computer Science, vol. 4389, pp. 221–238. Springer-Verlag (2007), invited chapter
19. Paes, R., Carvalho, G., Lucena, C., Alencar, P., Almeida, H., Silva, V.: Specifying laws in open multi-agent systems. In: In Agents, Norms and Institutions for Regulated Multiagent Systems - ANIREM (2005)
20. Pokahr, A., Braubach, L., Lamersdorf, W.: Jadex: A bdi reasoning engine. In: R. Bordini, M. Dastani, J.D., Seghrouchni, A.E.F. (eds.) Multi-Agent Programming. pp. 149–174. Springer Science+Business Media Inc., USA (9 2005), book chapter
21. Santos, J., Zahn, J., Silvestre, E., Silva, V.T., Vasconcelos, W.W.M.P.D.: Detection and resolution of normative conflicts in multi-agent systems: a literature survey. AAMAS **31**(6), 1236–1282 (4 2017)
22. Serban, I., Sordoni, A., Bengio, Y., Courville, A., Pineau, J.: Hierarchical neural network generative models for movie dialogues. ArXiv e-prints **abs/1507.04808** (July 2015)
23. von Wright, G.H.: Deontic logic. Mind **60**(237), 1–15 (1951)