Developing Argumentation Dialogues for Open Multi-Agent Systems

Bas Testerink & Floris Bex*

Utrecht University
{B.J.G.Testerink,F.J.Bex}@uu.nl

Abstract. Dialogue systems attempt to capture aspects of multi-agent communication with the aim of understanding, improving, and automatically recreating such communication. Though there is ample research into the formal aspects of dialogue games for argumentation, actual software and development tools that allow for the deployment of open, multi-agent dialogue environments are lacking. This demo presents the first steps towards a development framework for an open multi-agent system in which agents can engage in peer-to-peer argumentation dialogues.

Keywords: Open Multi-Agent Systems, Argumentation Dialogues, Agent-Oriented Programming

1 Introduction

In [10], we present a generic framework for expressing argumentation dialogues in open multi-agent systems. One of our motivations is that we aim to make the development of multi-agent argument dialogue systems more accessible and thus increase the deployment of such systems in realistic, large-scale settings. As a case-in-point, we are currently developing a prototype multi-agent system for the Dutch National Police [4], where various agents work together to build a case regarding internet trade fraud (e.g. scammers on eBay or fake online stores). We have agents that interact with human users (victims, police detectives), agents that exchange information with external services (banks, trade sites) and agents that automatically combine and reason with information from these different sources. Furthermore, the agents communicate peer-to-peer, that is, there is no governing middleware or entity that checks the communication. We have opted for this distributed, open multi-agent system because (i) the police uses strict privacy policies that make it undesirable to centrally gather and reason with data; and (ii) not all participating agents in the system are known beforehand – for instance, the human users that file online complaints are unknown.

As our main technique for reasoning we use argumentation [1] this fits the legal reasoning in the trade fraud cases very well and has as added benefit that

* `https://git.science.uu.nl/B.J.G.Testerink/OO2APL-P2PArgumentationDialogDemo` contains a video demo, a working demo and the source code, please see `readme.md`

we can make discussions among agents more transparent to human users. Communication among the agents is therefore governed by argumentation dialogue protocols, which allow us to determine whether a specific message from one agent to another is legal in the context of the dialogue so far.

In [10] we present our formal foundation for peer-to-peer argumentation protocol specifications based on reusable elements called *dialogue templates* [5]. In this paper we demonstrate a practical realization of this principle which is a development framework that supports programming protocols and protocol-interpreting agents. Our framework only requires programming knowledge of Java. Agents that use our modules always have access to the state of dialogue according to the protocol, based on the messages that they themselves sent and received. This view can be exported as a JSON object, so that the decision logic on what to do with that view (i.e., determining whether to utter a locution, and if so, what the next locution to utter is) can be implemented in another language if so desired.

Our framework is open-source and we hope that it stimulates the development of argumentation dialogue systems or may serve as an example to other developers. In this demonstration we present four main contributions: 1) the ability to develop protocols, 2) the agent modules that interpret a protocol to automatically determine which messages are legal and how they ought to be interpreted, 3) an example multi-agent system that can re-enact dialogues and 4) a visualization of the view of an agent on a dialogue, given the protocol.

In Section 2 we discuss the relevant terminology to understand the demo. Section 3.1 discusses the approach towards implementing protocols. Section 3.2 describes the example multi-agent system for dialogue re-enactment. Finally, 3.3 goes into detail on our example scenario and the visualization. Section 4 discusses related and future work and concludes the paper.

2 Background

Argumentation dialogues [7] consist of a coherent series of locutions, or utterances, made by the participants in the dialogue. Such multi-agent dialogues are governed by dialogue protocols, which describe the permitted locutions, how and when the dialogue starts and ends, how locutions commit the players to certain claims and how locutions may be combined into exchanges. Take, for example, one of the possible protocols for persuasion dialogues given in [7]. The protocol mandates that a dialogue starts when one agent, the proponent, makes a *claim* P, committing it to P. The other agent, the opponent, can either challenge this claim by uttering *why* P, which can in turn be countered by the proponent either retracting its commitment to P or by providing a supporting argument P *since* Q. A claim P can also be attacked by providing a counterargument $\neg P$ *since* Q. Furthermore, any argument of the form R *since* S made by either party can be countered by the other party by giving an argument $\neg R$ *since* T or $\neg S$ *since* T. The dialogue terminates when either the proponent *retracts* its original claim P or when the opponent *concedes* proponent's original claim P.

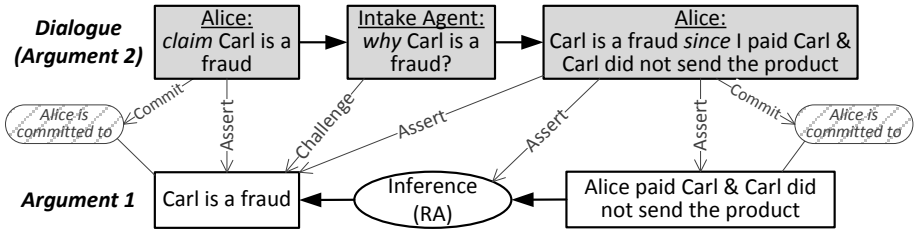

Fig. 1. Example graph combining argument 1 and 2.

For argumentation dialogues it is important to distinguish between argument 1 – arguments as a static structure of premises that are reasons for or against conclusions (as in, he prepared an argument) – and argument 2 – arguments as debates or dialogues (as in, they had an argument). The idea is that during a dialogue (argument 2) the participants construct and navigate an underlying argument 1 structure of claims and reasons [8]. For an example, consider Fig. 1. At the top (grey boxes), a simple dialogue in which Alice claims that Carl is a fraud is rendered, where the arrows indicate the flow of the dialogue. At the bottom (white boxes), the corresponding argument 1 is rendered: from the premise 'Alice paid Carl & Carl did not send the product' it is inferred that 'Carl is a fraud'. The link between argument 1 and 2 stems from speech act theory, where a speech act is a locution ('Alice says that Carl is a fraud'), but also an illocutionary act which consists of the illocutionary force, that is, the intention of uttering a locution (Intake Agent says '*why* Carl is a fraud?' with the intention of challenging that 'Carl is a fraud') and the propositional content ('Carl is a fraud'). Note that the same locution can encompass multiple illocutionary acts: by saying 'Carl is a fraud' Alice not only asserts that 'Carl is a fraud' but also commits herself to this proposition (cf. the small rounded box in Fig. 1).

Two core formal concepts in [10], on which our implementation is based, are dialogue graphs and dialogue templates. A dialogue graph captures the state of the dialogue from the point of view of one of the agents. In our implementation, this graph represents argument 1, argument 2 and commitments (cf. Fig. 1). The argument graphs in our implementation are slightly adapted versions of AIF argument graphs [8], which are based on the underlying AIF ontology specification [3]. This ontology places a distinction between information nodes or I-nodes, to represent locutions or propositions, and scheme nodes or S-nodes, to denote general patterns of reasoning or the relations between I-nodes. Subclasses of S-nodes are RA (Rule Application) nodes and CA (conflict application) nodes. We further add commitment nodes indicating the commitment of an agent to an I-node.

Dialogue templates define how to update the graph when receiving and sending locutions. They are used to represent dialogue protocol rules, as they contain for each type of locution that is sent or received the possible replies (transitions to other locutions). For example, one template says that if a *why P* message is received, the argument 2 graph is changed to include this locution, the argu-

ment 1 graph remains unchanged, and a P *since* Q message may be sent back. Another template says that sending a P *since* Q message not only updates argument 2 but also updates the argument 1 graph to include Q and the inference (RA-node) between Q and P, and also commits the sender to P and Q.

3 Software Overview and Demonstration

3.1 Programming Protocols

Our framework provides preprogrammed classes that help a developer to specify dialogue graphs and templates. We preprogrammed the following types of nodes in dialogue graphs: `Argument2Node`'s represent locutions that were uttered and edges between these nodes represent transitions from one locution to the next. AIF I- and S-nodes are implemented as `INode`'s and `SNodes`'s. `CommitmentNode`'s track which agent is committed to what propositions. `EffectNode`'s track the illocutionary force of locutions, that is, what the effect of a locution is on the argument 1 graph or the commitments. The default dialogue graph implementation can be extended for added functionalities. This is demonstrated with our demo protocol where the `DemoDialogGraph` maintains a mandate for each agent what they may utter to what other agent.

Templates can be made by supplying supplying three functions called the guard, response and argument 1 update function. The guard of the template tells given a dialogue graph and a new argument 2 node whether the template is applicable. The response function returns, given a dialogue graph and a new locution, the argument 2 nodes from the dialogue graph to which the new locution is a direct response. The argument 1 update is a function that accepts a dialogue graph and a new argument 2 node. The template class has a preprogrammed apply-method that for a new message checks the guard, and if the guard passes, applies the response function and argument 1 update. Any changes that were made during the execution of the argument 1 update are automatically recorded. After the function is applied, the template will create an effect node that connects the new message to the changes that were made while executing the argument 1 update function. A protocol itself is simply a set of templates as demonstrated in the `demoProtocol` package.

Finally, we preprogrammed classes for the development of Java-based agents that interact using argumentation dialogue protocols. For this we built on top of OO2APL [6][1]. When an agent receives a message it will check if the sender was allowed to sent the message by checking whether any of the templates applies given the current dialogue graph. If not, then the agent immediately replies with a bounce message to the sender. If there was an applicable template, then that template updates the dialogue graph. The agent can send a message with a preprogrammed send method which automatically does the entire process again, but for the message to be sent. If the agent tries to illegally send a message it automatically receives an internal error.

[1] `https://git.science.uu.nl/OO2APL/`

3.2 Re-enacting Dialogues

Our demonstration comes with a dialogue reader which is a multi-agent system that can re-enact a dialogue. There are different ways in which the reader can be used. When developing a protocol, one can write manually dialogues (similar to user stories) and see whether the protocol builds the dialogue graph as intended. For instance in our example scenario we do not want that the intake agent commits itself in the dialogues with complainants to the fact that someone is a fraud[2]. Another usage is to re-enact the dialogue given a log to trace back when certain agreements where reached or where some agent violated the protocol.

There are two mandatory input files for the dialogue reader. The first one is the invitations file, that tells the system who recruited who into the argumentation session. The second input file is the script/log of the locutions that were uttered. On the Java level the reader further requires a parser from strings (the locutions in the locutions file) to the locution objects that represent them, and the constructor of the protocol that is to be used. The reader then reads all the files, re-enacts the dialogue, and outputs the dialogue graphs of the agents as JSON. The re-enactment is done by instantiating `DialogReaderAgent` agents. Such an agent's decision logic when it receives a message is to simply prompt the next agent to send the next locution in the script.

Note that this dialogue reader multi-agent system can be used as a starting point: the agents in have a method called `handleReceivedArgumentationMessage` of which the body can be replaced by a different decision logic on how to respond. Thus, multi-agent systems that include agents with their own knowledge bases and decision logics can be easily developed.

3.3 Example Dialogue and Visualization

The dialogue reader creates JSON objects that represent the dialogue graphs of the dialogue participants, which are visualized on a web page[3]. The graph is split into five components; the dialogue itself, the argument 2 graph, the argument 1 graph, the illocutionary force inspector and the AIF graph.

Our demonstration uses an example that is based on our project at the Dutch National Police, and also strongly resembles the current modus operandi for handling for instance online trade fraud cases. The scenario is that complainants can file their complaints by interacting with an intake agent and a detective can interact with an analyst agent. When the detective asks the analyst agent a question, it discusses the question with the intake agent and another agent that may requisition information from third parties. Our demo thus contains agents that represent humans, third parties and automated agents. The human agents in our demo are the complainants Alice and Bob, who file complaints that accuse

[2] With online trading fraud, your complaint can have all the signs of a fraud case without being a fraud case. Roughly speaking, if someone just forgot to send an order once, then the police will not immediately arrest that person for fraud.

[3] see `/visualizer/visualizer.html` in the demo repository

some person named Carl of fraud, and a detective, who asks if Carl is a suspected fraudster. The third parties are the Dutch trading website Marktplaats, a bank, and the general administration system from municipalities called GBA. The automated agents stand in between the other agents. A requisition agent can demand information from the third parties, an intake agent interacts with complainants and an analyst agent interacts with detectives. The requisition, intake and analyst agent can also interact with each other to discuss topics.

For a visualization of the agents and who is talking to who, see the top part of the demo page or 'connectivityGraph.png' in the screen shot directory. The agents in our demonstration discuss the topic of whether Carl is a fraud. The full transcript of the discussion that the agents have can be found in the 'dialogues/intake/locutions.csv' file. We can click on an agent's icon in order to inspect its view on the dialogue. As an example we will go through the view of Alice who files a complaint with the intake agent. In the dialogue tab (or 'dialogue.png') we can see a timeline with events as observed by Alice. Here, we only see the messages that Alice sent and received to and from the intake agent. Initially, Alice claims that Carl is a fraud. The agent replies with a why-challenge to trigger Alice into defending her claim. She then replies with the argument that since she paid and Carl did not sent her package, it must be the case that Carl has scammed her. To the intake agent it is not obvious why Alice thinks that the package has not been sent. It therefore challenges this claim with another why locution. Alice defends her claim by stating that she waited but that the package was not delivered. The intake agent then concedes the point that the package was probably not sent by Carl. Alice has nothing else to say so she skips. The intake agent then asks whether Carl sent any pictures of an ID card and whether Carl gave his address. Both questions are answered affirmatively by Alice and the intake dialogue ends.

In the argument 2 tab, we can click on the cluster 'Alice, Intake agent' to show the locutions that were uttered and the response relation among them (alternatively, see 'argument2.png'). From this view we can see that the protocol interpreted the dialogue between Alice and the intake agent as three coherent sub-dialogues. There is a connected graph concerning the topic of whether Alice thinks Carl is a fraud, a connected graph about whether Carl sent picture of an ID card, and a connected graph about whether Carl gave his address.

The argument 1 tab (alternatively, see 'argument1.png') shows the effect of the dialogue in terms of argument 1, that is, the propositions that were asserted or committed to and the inferences that were made. We can see, for instance, that an inference was made from 'Alice paid' and 'The package was not sent' to 'Carl is a fraud'. We can also see that the agent does not commit to this inference, nor to the conclusion. This is a desired outcome, since the intake agent ought not commit itself to propositions regarding criminal facts.

In the illocutionary force tab we can see for each locution how it modified the argument 1 graph. For instance we can see that the concede locution (click on locution 6) from the intake agent that referred to the proposition that Alice's package was not sent, commits the agent to that proposition (alternatively, see

'illocutionaryForce.png' in the screenshot directory). Finally, the AIF tab shows for inference in the argument 1 graph the applied argument scheme and how the elements are connected to the AIF ontology (cf. [3]). We use in this demonstration only the schemes 'apply defeasible rule' for inference and 'conflict by negation' proposition conflicts.

When we look at the dialogue from the detective's view, then we only see it ask the analyst whether Carl is a fraud, and immediately see the analyst's response that this is not the case. The detective does not see the entire discussion that went on between the analyst, intake and requisition agents about this question, nor is he bothered with the filed complaints. Even though the intake agent is initially convinced that Carl is a fraud, based on the complaints, the analyst persuades the intake agent that actually Carl just did not deliver as an exception to generally bona fide behaviour. Notice that Alice and Bob argue that Carl is a fraud differently from the intake agent. Even though the intake agent uses different argumentation rules than Alice and Bob, it can still have meaningful and useful dialogues with them.

4 Conclusion

In this paper we have discussed our development framework for programming argumentation dialogue protocols that are suitable for peer-to-peer settings. We demonstrated this framework with an example that was taken from an ongoing research project with the Dutch National Police. The framework allows a programmer to make a protocol by specifying the dialogue templates. The software also provides a multi-agent system that can read transcripts of dialogues and produces the accompanying dialogue graphs for each participant. The dialogue graphs are exported as JSON objects, which makes the software also usable to connect to other technologies through a web-interface. Finally, we demonstrated a web-based visualization of the dialogue graph JSON objects so that a developer can inspect the graphs in a more human-readable manner.

Closely related to our framework is the Dialogue Game Execution Platform (DGEP) [2], which is also a tool that helps in developing argumentation dialogue protocols. The main difference with our framework is that DGEP is not an open multi-agent framework but rather a middleware platform that, given a locution, outputs the legal moves. Our framework can be used to program agents that interact with the DGEP platform if desired, although this is not strictly necessary since our agents can interpret protocol specifications themselves.

Future work concerns the research that serves as the basis for this demo. We have formalized peer-to-peer protocols [10] but certain aspects of the demo, such as the connection between argument 1 and 2, have not yet been fully worked out. A further aim is to be able to automatically compile the DGDL specifications [11] of the many argumentation dialogue protocols that are available to a protocol in the presented development framework. Another major point is the formalization and implementation of different agent argumentation strategies.

Finally, with respect to intelligent interfaces, such as the intake agent, it is desirable that humans can interact with such agents in a natural way - while some form-based interaction with pre-set choices is fine, there are also situations in which we would want the human to be able to simply type in natural language text which is then parsed and used by the automated intake agent. In our project, we have taken first steps to evaluate the possibilities of such information extraction [9] and aim to further develop this in the context of the types of dialogue explained in this demonstration.

Acknowledgments

This research is part of the project *Intelligence Amplification for Cybercrime*, which has been funded by the Dutch National Police Innovation Programme.

References

1. T. J. Bench-Capon and P. E. Dunne. Argumentation in artificial intelligence. *Artificial intelligence*, 171(10-15):619–641, 2007.
2. F. Bex, J. Lawrence, and C. Reed. Generalising argument dialogue with the dialogue game execution platform. In *Proceedings of COMMA 2014*, volume 266 of *Frontiers in Artificial Intelligence and Applications*, pages 141 – 152. IOS Press, 2014.
3. F. Bex, S. Modgil, H. Prakken, and C. Reed. On logical specifications of the argument interchange format. *Journal of Logic and Computation*, 23(5):951–989, 2012.
4. F. Bex, J. Peters, and B. Testerink. A.I. for online criminal complaints: From natural dialogues to structured scenarios. In *Workshop A.I. for Justice - Proceedings of ECAI 2016*, 2016.
5. F. Bex and C. Reed. Dialogue templates for automatic argument processing. In *Proceedings of COMMA 2012*, volume 245 of *Frontiers in Artificial Intelligence and Applications*, pages 366–377. IOS Press, 2012.
6. M. Dastani and B. Testerink. From multi-agent programming to object oriented design patterns. In *Engineering Multi-Agent Systems*, pages 204–226. Springer International Publishing, 2014.
7. H. Prakken. Formal systems for persuasion dialogue. *The Knowledge Engineering Review*, (21):163–188, 2006.
8. C. Reed, S. Wells, K. Budzynska, and J. Devereux. Building arguments with argumentation: the role of illocutionary force in computational models of argument. In *Proceedings of COMMA 2010.*, volume 216 of *Frontiers in Artificial Intelligence and Applications*, pages 415–426, 2010.
9. M. Schraagen, M. Brinkhuis, and F. Bex. Evaluation of Named Entity Recognition in Dutch online criminal complaints. *Computational Linguistics in the Netherlands Journal*, 7, 2017. *To appear.*
10. B. Testerink and F. Bex. Specifications for peer-to-peer argumentation dialogues. In *Proceedings of PRIMA 2017*, Lecture Notes in Artificial Intelligence. Springer, 2017. *To appear.*
11. S. Wells and C. Reed. A domain specific language for describing diverse systems of dialogue. *Journal of Applied Logic*, 10(4):309–329, 2012.