A Practical Framework for General Dialogue-based Bilateral Interactions

Seyed Ali Hosseini[1], David Minarsch[1], and Marco Favorito[1,2]

[1] Fetch.ai, UK
[2] Sapienza University of Rome, Italy
{ali.hosseini, david.minarsch, marco.favorito}@fetch.ai

Abstract. For autonomous agents and services to cooperate and interact in multi-agent environments they require well-defined protocols. A multitude of protocol languages for multi-agent systems have been proposed in the past, but they have mostly remained theoretical or have limited prototypical implementations. This work proposes a practical realisation of a general framework for defining dialogue-based bilateral interaction protocols which supports arbitrary agent-based interactions. Crucially, this work is tightly integrated with a modern framework for the creation of autonomous agents and multi-agent systems, making it possible to go from protocols' specification to their implementation and usage by agents, and enables evaluation of protocols' effectiveness and applicability in real-world use cases.

Keywords: Interaction Protocols · Dialogues · General Protocols.

1 Introduction

Motivation Multi-Agent System (MAS) is recognised as a promising paradigm for decentralised and ubiquitous computing that involves embedded and distributed devices interacting with each other [30]. The increasing complexity and scale of these systems necessitates the development of abstractions and tools that simplify their development and deployment. This demand gives rise to *interaction protocols* as a key mechanism that enables cooperation amongst agents while recognising their individuality in a decentralised environment and accommodating their competing interests [14].

Interaction protocols are a useful abstraction which not only help the process of agent design by limiting the space of all possible states and actions in specific interactions, but also enable analysis of agent-based systems to assess specific properties, for example checking whether a system could arrive at a deadlock [49].

There are a multitude of protocol languages in the MAS literature based on various formal abstractions and mathematical constructs, for example, based on UML [25], state machines [49,22], trace expressions [18,13], and session types [52]. However, to the best of our knowledge, they either remain theoretical (e.g. [13]), or have limited prototypical implementations (e.g. [6,49,52]). As a result, the applicability and effectiveness of interaction protocol systems, as part of agent-based solutions to real-world problems, are not fully explored.

Requirements There are two fundamental requirements that guide this work. Firstly, (**Req. 1**) we are interested in environments which are fully *decentralised*, a property often considered integral to MAS itself [14]. Decentralisation refers to the absence of central authorities that imperatively control aspects of the system (e.g. decision making, communication, authorisation, coordination, etc.). In decentralised environments, interactions are primarily peer-to-peer, without reliance on third-party facilitators. This assumption immediately distinguishes this work from proposals such as [6,21] in which interaction protocols are enforced via mediators and middleware.

Secondly, (**Req. 2**) the framework must be practically realised by an implementation that is accessible, enabling its application and evaluation in real-world use cases. We believe that a lack of attention to implementation leads to unexplored aspects of the system design, or in some cases major oversights. For example, a number of protocols in the literature are defined with reference to private elements of an agent (e.g. mental state) [3]. However, it is not entirely clear how such protocols may be implemented and enforced in practice under standard MAS assumptions.

Contribution and Structure This work resides in a larger body of work by the authors ([32,40,35,33,34]) to bring agent technologies to production by taking advantage of strategic integration with distributed ledger technologies (DLTs) [47].

In this paper, we propose a general framework for dialogue-based bilateral interactions that use protocols to govern the behaviour of interaction participants. The framework is formally defined and its implementation facilitates its application in real-world problems. The implementation is integrated into the *AEA framework*; a modern framework for the development and deployment of agents [34].[3] This allows interactions and protocols to be specified then implemented and used by agents.

After informally describing the setting and highlighting key design issues in Section 2, we formalise the framework in Section 3 and instantiate it to capture a specific interaction: bilateral negotiation. We then shift our attention in Section 4 to implementation, discussing the main components of the framework and the major implementation issues involved. Section 5 provides a discussion of related works. Finally, Section 6 concludes and outlines future work.

2 Dialogue-based Bilateral Interactions

We define *Bilateral interactions* as well-defined high-level interactions between exactly two *players* that serve a clear purpose. In the kinds of environment we focus on, players could be agents, services or humans. Some example are, bilateral negotiations [4], state channels [31], information-seeking [27] and HTTP request/responses.

The decentralised nature of multi-agent systems, the autonomy of agents, and the heterogeneity of their designs have all contributed to the established practice of modelling events and interactions amongst agents as *messages* [15,50,9,20,12]. A *dialogue* then structures and encapsulates a series of messages exchanged as part of a single interaction [38]. There are many dialogue-based approaches [2,4,39,11,37] to various types of interactions (e.g. negotiation, persuasion, inquiry) [46].

[3] The AEA framework's repository can be found at https://github.com/fetchai/agents-aea

The peer-to-peer nature of communication in fully decentralised environments (see **Req. 1**) motivates our focus on *bilateral* interactions. Of course¸ it also simplifies the system design and helps us focus our attention on achieving an end-to-end solution in line with **Req. 2**. Moreover, bilateral primitives can later be used as the foundation of multilateral extensions to this work.

Any interaction serves a particular (set of) goal(s), called the *interaction's goal(s)*. This is what all participant's aim to achieve by taking part in the interaction. For instance, the goal of a negotiation is dividing scarce resources amongst multiple parties [46]. All players in an interaction also have *personal goals* which may not necessarily be the same as the interaction's goal. For instance, in negotiations, each player aims to maximise its share of the resources.

2.1 Protocols

Protocols specify the bounds within which players in an interaction may operate to ensure an interaction's goal(s) are fulfilled [19,28].

A key issue to consider, when designing a concrete implementation of agent interactions, is how players' adherence to protocols is verified. In particular, anyone who is observing an interaction must be able to confirm whether or not the players' actions conform to the protocol. This idea underpins proposals such as [21,6] which introduce middle-layer *moderator* agents that coordinate communications and enforce protocols on players. However, in decentralised environments, where the absence of such middleware is entailed, this responsibility may be reliably assigned only to the players participating in the interaction themselves.

A consequence of the above is that protocols are restricted, in design, to making reference only to public elements of an interaction [44]. For example, proposals (such as [3]) which make references, either in syntax or semantics of their protocols, to e.g. agents' mental states, private strategies, etc., are in our experience not straightforward to implement under standard MAS assumptions. Therefore, in this framework, protocols can only be defined with reference to public elements of an interaction and protocol adherence is verified by the players of the interaction themselves.

Another key design issue to consider is the assumption of (a)synchronicity in message delivery [23]. Many proposals for interaction protocols in the literature assume synchronous communication for simplicity [25,22]. However, we argue that the uncertainty associated with the communication infrastructure, coupled with agents' autonomy, and the possibility of agents engaging in parallel interactions make synchronous communication, which blocks some or all other agent processes, detrimental to the continuity and successful operation of a decentralised MAS. Therefore, in this framework, we do not take the synchronicity assumption on board and address the problems it causes on the framework level.

3 Framework

We now give a formal description of a general framework for bilateral dialogue-based interactions and then provide an instantiation that captures bilateral negotiations.

3.1 Bilateral Interactions

The environment is inhabited by players. In practice, a player might be an agent, service, human, or other entity.

Definition 1. ***[Player]*** *A finite set $\mathcal{A}$ is defined where each $a \in \mathcal{A}$ is a* player.

A role is a logical actor in the context of an interaction identified by a name (e.g. bidder, seller). Players participating in a dialogue are assigned roles (see Definition 11). This is how protocols apply to specific players in a dialogue instance.

Definition 2. ***[Role]*** *A set $\mathcal{R}$ is defined where any $r \in \mathcal{R}$ is a* role *and $|\mathcal{R}| \in \{1, 2\}$.*

$|\mathcal{R}| = 2$ means each player has a distinct role and $|\mathcal{R}| = 1$ indicates that both players have identical roles.

Dialogues progress by players exchanging messages. Before defining a message, we define the notions of speech-act (see [7]) and their reply structure:

Definition 3. ***[Speech-act]*** *A set $\mathcal{S}$ of speech-acts $\{s_1, \ldots, s_n\}$ is defined where each speech-act s_i is of the form $\mathsf{P}(\mathsf{c}_1, \ldots, \mathsf{c}_n)$ where P is an element of a set $\mathcal{P}$ of* performatives*, and $\mathsf{c}_1, \ldots, \mathsf{c}_n$ is a sequence of* contents*.*

Example 1. Examples of speech-acts are *inform*(ϕ), *offer*(ψ), *commit*(ω), *request*$(\rho_1, \ldots, \rho_4)$ where *inform*, *offer*, *commit*, *request* are performatives and ϕ, ψ, ω (some information) and $\rho_1, \ldots, \rho_4$ (some resources) are contents.

Definition 4. ***[Reply]*** *A function* $\texttt{Reply} : \mathcal{S} \longrightarrow 2^{\mathcal{S}}$ *specifies the valid* replies *to each speech-act.*

Example 2. Let $\mathcal{S} = \{req(\phi), res(\psi)\}$ respectively represent a request and response involving some information ϕ and ψ. Then

- $\texttt{Reply}(req(\phi)) = \{res(\psi)\}$
- $\texttt{Reply}(res(\psi)) = \emptyset$

Definition 5. ***[Message]*** *The set $\mathcal{M}$ of messages is defined as $\{\langle id, pl, sa, ta\rangle \mid id \in \mathbb{N},\ pl \in \mathcal{A},\ sa \in \mathcal{S},\ ta \in \mathbb{N}\}$ such that the four elements of a message m are respectively denoted by:*

- $\texttt{id}(m)$*: the* identifier *of the message*
- $\texttt{player}(m)$*: the* player *sending the message*
- $\texttt{speech-act}(m)$*: the* speech-act *in the message*
- $\texttt{target}(m)$*: the* target *of the message (i.e. the id of the message it replies to)*

Note how each message targets another in a dialogue. This is how protocols (defined later) use the notion of reply.

Definition 6. ***[Dialogue]*** *The set of* dialogues*, denoted $\mathcal{D}$, is the set of all finite sequences $m_1, \ldots, m_n$ from $\mathcal{M}$ such that each i^{th} message in the sequence has identifier i. Given a dialogue d, the set of players participating in d is denoted by $\mathcal{A}^d$, and for every message $m_i \in d$, $\texttt{player}(m_i) \in \mathcal{A}^d$.*

A dialogue is thus a sequence of messages exchanged between a set of participating players, as viewed by any player. A dialogue is bilateral if messages are exchanged between exactly two players:

Definition 7. ***[Bilateral Dialogue]*** *A dialogue d is* bilateral *iff $\mathcal{A}^d = \{a, b\}$ for some players $a, b \in \mathcal{A}$ where $a \neq b$.*

Remark 1. The above definitions for dialogues specify the structure that each dialogue participant maintains and not a construct globally shared between them. Due to the asynchronous message exchange, the order of messages in dialogues on either side may be different. This is discussed more in Section 4.1.

Notation 8 *Let $d = m_1, \ldots, m_i, \ldots, m_n$ be some arbitrary dialogue. Then:*

- *d_0 represents an empty dialogue.*
- *d_i represents $m_1, \ldots, m_i$.*
- *d, m represents the continuation of dialogue d with message m.*
- *$d_{n'}$ is a* sub-dialogue *of d_n iff $n' \leq n$ and the first n' messages in d_n is the same as those in $d_{n'}$.*

An interaction must specify how a dialogue commences by describing the speech-act(s) which can be used to start a dialogue:

Definition 9. ***[Commencement]*** *A non-empty subset $\mathcal{C} \subseteq \mathcal{S}$ of speech-acts $\mathcal{S}$ is defined as* initial. *Any dialogue d_n is* commenced *at 1 and $\texttt{speech-act}(m_1) \in \mathcal{C}$.*

In some cases, it is useful for an interaction to define dialogue termination criteria so the participating agents have prior agreement as to when their dialogue is terminated. These criteria are defined in our framework via speech-acts.

Definition 10. ***[Termination]*** *A subset $\mathcal{T} \subseteq \mathcal{S}$ of speech-acts $\mathcal{S}$ is defined as* terminal. *A dialogue d_n is* terminated *at n, iff $\texttt{speech-act}(m_n) \in \mathcal{T}$ and it is not the case that d is terminated at an earlier point $n' < n$.*

Note that if $\mathcal{T} = \emptyset$, an interaction does not impose dialogue termination, and it is up to the participants to decide the criteria for recognising when a dialogue is terminated (e.g. after a period of inactivity).

The role of a player in a dialogue is defined as follows:

Definition 11. ***[Role Assignment]*** *The partial function $\texttt{R} : \mathcal{A} \times \mathcal{M} \longmapsto \mathcal{R}$ describes the role of any player a in a dialogue whose first message is m, i.e. where*

- $\texttt{id}(m) = 1$
- $\texttt{player}(m) \in \{a, a'\}$ *where $\{a, a'\} \in \mathcal{A}^d$ for some dialogue d*
- $\texttt{speech-act}(m) \in \mathcal{C}$
- $\texttt{target}(m) = 0$ *(see R_6 in Definition 15)*

If m is not an initial message or $a \notin \mathcal{A}^d$, then $\texttt{R}(a, m) = \texttt{undefined}$.

Example 3. A simple request/response interaction in which roles are defined as $\mathcal{R} = \{inquirer, respondent\}$, speech-acts and reply structure as those in Example 2, and where $\mathcal{C} = \{req(\phi)\}$, defines the assignment of roles to players as follows:

$$\mathtt{R}(a, m) = \begin{cases} inquirer & \text{iff } \mathtt{player}(m) = a \\ respondent & \text{otherwise} \end{cases}$$

A turn-taking function defines how turns shift in a dialogue:

Definition 12. ***[Turn-Taking]*** *A* turn-taking *function is* $\mathtt{Turn} : \mathcal{D} \longrightarrow 2^{\mathcal{A}}$ *where* $\mathtt{Turn}(d) \in \mathcal{A}^d$*, specifying the player(s) who have the right to send the next message in a dialogue.*

We now define the notion of well-formed dialogues:

Definition 13. ***[Well-Formed Dialogue]*** *A subset of dialogues* $\mathcal{D}^w \subseteq \mathcal{D}$ *are* well-formed *with the condition that a)* d_0 *is always in* $\mathcal{D}^w$*, and b) iff* $d_n \in \mathcal{D}^w$ *so are all of* d_n*'s sub-dialogues* $d_{n'}$ *where* $n' < n$*.*

Any terminated well-formed dialogue has an outcome that determines the state in which the dialogue is terminated:

Definition 14. ***[Outcome]*** *A non-empty finite set* $\mathcal{O}$ *is defined where each* $o \in \mathcal{O}$ *is a dialogue* outcome*. A partial function* $\mathtt{Outcome} : \mathcal{D}^w \longrightarrow \mathcal{O}$ *maps any terminated dialogue* d *to an outcome and* $\mathtt{Outcome}(d) = \mathtt{undefined}$ *if* d *is not terminated.*

Protocols define the legality of a message with reference to a dialogue:

Definition 15. ***[Protocol]*** *A* protocol *is a labelling function* $\mathtt{Legal} : \mathcal{D}^w \times \mathcal{M} \longrightarrow \{\mathtt{True}, \mathtt{False}\}$ *which satisfies protocol rules* $\mathrm{R}_1 - \mathrm{R}_6$ *below:*
For any $d_n \in \mathcal{D}^w$ *and* $m \in d_n$*,* $\mathtt{Legal}(d_n, m) = \mathtt{True}$ *iff:*

- R_1*:* $\mathtt{player}(m) \in \mathtt{Turn}(d_n)$
- R_2*:* $\mathtt{speech\text{-}act}(m) \in \mathcal{S}$.
- R_3*:* $\mathtt{id}(m) = n + 1$
- R_4*:* $\mathtt{target}(m) = 0$ *iff* $\mathtt{id}(m) = 1$*, otherwise* $1 \leq \mathtt{target}(m) < \mathtt{id}(m)$.
- R_5*: if* m *replies to* $m' \in d_n$*, then* $\mathtt{speech\text{-}act}(m) \in \mathtt{Reply}(\mathtt{speech\text{-}act}(m'))$.
- R_6*:* d_n *is not terminated at* n*.*

Protocol rules R_1 and R_2 respectively ensure that messages are sent by the 'right' players in a dialogue, as specified by the turn-taking function, and that they have the correct speech-acts (per Definition 3). Protocol rule R_3 ensures that each message is correctly placed right after the last message in the dialogue. R_4 and R_5 regulate replies by stating that only the first message replies to no other message ($\mathtt{target}(m_1) = 0$), and any other message targets another one in the dialogue while respecting the reply structure of speech-acts (Definition 4). Finally, R_6 states that once a dialogue is terminated, no other message can be legally added.

Together, the above rules define a lower bound on message legality. Of course, additional rules may be defined in specific interactions, for example, in the protocol that will be described in the next section.

Definition 16. ***[Well-Formed Dialogues Against Protocols]*** *A dialogue d_n is* well-formed against a protocol $\texttt{Legal}$*, iff* $\texttt{Legal}(d_{n-1}, m_n) = \texttt{True}$.

Remark 2. Note that dialogues, well-formed under any protocol, are recursively well-formed on their sub-dialogues due to Definition 13.

Proposition 1. *Let $\mathcal{I}$ be an interaction dialogue system where $s \in \mathcal{C}$ and $\nexists s' \in \mathcal{S}$ such that $s \in \texttt{Reply}(s')$. There is no well-formed dialogue d in $\mathcal{I}$ that contains a message m where* $\texttt{speech-act}(m) = s$.

An interaction dialogue system can now be defined:

Definition 17. ***[Interaction Dialogue System]*** *An Interaction Dialogue System is a tuple $\langle \mathcal{R}, \mathcal{S}, \texttt{Reply}, \mathcal{C}, \mathcal{T}, \texttt{R}, \texttt{Turn}, \texttt{Outcome}, \texttt{Legal} \rangle$ where $\mathcal{R}$ is a set of roles, $\mathcal{S}$ is a set of speech-acts,* $\texttt{Reply}$ *is a reply function, $\mathcal{C}$ is the initial speech-acts, $\mathcal{T}$ is the terminal speech-acts,* $\texttt{R}$ *is a role assignment function,* $\texttt{Turn}$ *is a turn-taking function,* $\texttt{Outcome}$ *is an outcome function, and* $\texttt{Legal}$ *is a protocol.*

3.2 Framework Instantiated: Bilateral Negotiation

The framework presented above is abstract and needs to be instantiated to capture specific interactions.

A bilateral negotiation is an interaction between two agents that negotiate over a set of resources. There are many examples of bilateral negotiation systems in the literature [3,43,4]. Here, we present a simple two-party negotiation as an instance of the framework:

Definition 18. ***[Negotiation Dialogue System]*** *An* interaction dialogue system for bilateral negotiation *is an interaction dialogue system $\langle \mathcal{R}_n, \mathcal{S}_n, \texttt{Reply}_n, \mathcal{C}_n, \mathcal{T}_n, \texttt{R}_n, \texttt{Turn}_n, \texttt{Outcome}_n, \texttt{Legal}_n \rangle$ where:*

- ***(Negotiation Roles)*** *$\mathcal{R}_n = \{b, s\}$ where b stands for* buyer *and s* seller
- ***(Negotiation Speech-acts)*** *$\mathcal{S}_n$=$\{cfp(e), propose(e,p), accept(), decline()\}$ where e is a non-empty set of resources and $p \in \mathbb{R}_{\geqslant 0}$ denotes a price*
- ***(Negotiation Reply)***

$$\begin{aligned}
&\texttt{Reply}_n(cfp(e)) = \{propose(e,p), decline()\}\\
&\texttt{Reply}_n(propose(e,p)) = \{propose(e,p'), accept()\}\\
&\texttt{Reply}_n(accept()) = \texttt{Reply}(decline()) = \emptyset
\end{aligned}$$

- ***(Negotiation Commencement)*** *$\mathcal{C}_n = \{cfp(e)\}$*
- ***(Negotiation Termination)*** *$\mathcal{T}_n$=$\{accept(), decline()\}$*
- ***(Negotiation Role Assignment)*** *Let m be an initial message and $x \in \mathcal{A}$:*

$$\texttt{R}(x, m) = \begin{cases} b & \textit{iff } \texttt{player}(m) = x \\ s & \textit{otherwise} \end{cases}$$

– ***(Negotiation Turn-Taking)*** *Let $x, x' \in \mathcal{A}$ s.t. $x \neq x'$:*

$$\texttt{Turn}_n(d_i) = \begin{cases} \{x\} & \textit{iff } i \textit{ is even} \\ \{x'\} & \textit{otherwise} \end{cases}$$

– ***(Negotiation Outcome)*** $\mathcal{O}_n = \{\texttt{a-r}, \texttt{a-u}\}$ *where* `a-r` *stands for* `agreement-reached` *and* `a-u` *for* `agreement-unreached`*:*

$$\texttt{Outcome}_n(d_i) = \begin{cases} \texttt{a-r} & \textit{iff } \texttt{speech-act}(m_i) = accept() \\ \texttt{a-u} & \textit{otherwise} \end{cases}$$

– ***(Negotiation Protocol)*** $\texttt{Legal}_n : \mathcal{D}^w \times \mathcal{M} \longrightarrow \{\texttt{True}, \texttt{False}\}$ *is a protocol, that in addition to* $\mathrm{R}_1, \mathrm{R}_6$*, satisfies the negotiation rule* N_1 *below: For any* $d_i \in \mathcal{D}^w$ *and* $m \in d_i$, $\texttt{Legal}_n(d_i, m) = \texttt{True}$ *iff:*
  - N_1*: if* $\texttt{speech-act}(m) = propose(e, p)$ *or* $\texttt{speech-act}(m) = accept()$*, and* m *replies to* m' *then* $\texttt{player}(m) \neq \texttt{player}(m')$.

4 Implementation

In line with **Req. 2** in the introduction, we developed a technical implementation of the formalism we proposed in Section 3. In what follows, we describe its major components and highlight important implementation issues.

4.1 Practical Considerations

Asynchronisation Recall from Section 2 that in this work, we do not assume communication between agents to be synchronous. The asynchronisation means that the two dialogue structures (see Definition 6), held by each participant in a dialogue, may not necessarily be identical. Consider an example interaction with the following turn taking function:

$$\texttt{Turn}(d) = \{a, b\} \text{ for any } d, \text{ where } a, b \in \mathcal{A}^d$$

Let us assume a dialogue d_n between a and b. At point n, agent a sends message m while simultaneously b sends m' (note the turn-taking function essentially allows any participant to send a message at any point in the dialogue). After these moves, a's dialogue is d, m, m' and b's is d, m', m.

This discrepancy entails that in addition to the explicit dialogue structures held by each participant in a dialogue, there exists an *implicit* structure, to which no participant has access, that could represent the global state of the dialogue. This structure, unlike local dialogues, is not a sequence, rather a poset (i.e. partially ordered set) where only some of the messages are ordered and some are incomparable.

Definition 19. ***[Global Dialogues]*** *A global dialogue is a tuple $\langle M, \prec \rangle$ where $M \subset \mathcal{M}$ is a set of messages and $\prec$ is a partial order.*

Note that $m_i \prec m_j$ means m_i precedes m_j in both participants' dialogues, and having $m_i \not\prec m_j$ and $m_j \not\prec m_i$ means m_i and m_j are incomparable and thus ordered differently in the two dialogues.

In the implementation, this means that messages are not uniquely identified only by their *id* (e.g. when identifying the message m' that a message m replies to). Instead, *id* and *player* combined are used to uniquely identify each message. This is achieved in the implementation by splitting the set of non-zero integers into two mutually exclusive subsets of positive $\mathbb{Z}^+$ and negative $\mathbb{Z}^-$ numbers, assigning the former to the player starting the dialogue (let us call it a) and the latter to the other player (let us call it b). As a result, a continuously increments and b decrements the id of the messages they send (replacing R_3 in Definition 15). The second part of R_4 then is replaced with the condition that $\texttt{target}(m)$ is between the smallest negative and largest positive id excluding 0.

The incomparability of certain messages in an implicit global dialogue, and hence the discrepancy of orders between local dialogues is not, in and of itself, a problem. In some use cases, this incomparability does not matter, especially when taking into account that the reply mechanism, strictly enforced in protocols, can cover causal ordering (e.g. a speech-act s which is strictly required to be after s' can be defined as its reply). It is however an issue that must be considered when designing specific interactions.

Moreover, there are interaction designs with specific reply structures and turn-taking functions that guarantee messages in the global dialogue are totally ordered. These results are beyond the scope of this work and will be presented in future work on the properties of interaction designs.

Parallel Dialogues In practice, agents may be engaged in multiple interactions and dialogues at the same time. Therefore, an agent x who is having two simultaneous negotiations with an agent y, should be able to recognise the correct dialogues y's messages belong to. To address this, each dialogue in the implementation is assigned a reference by its two players. Thus, in addition to the four elements in Definition 5, each message includes a reference to the dialogue to which it belongs.

4.2 Protocol Specification

The decentralised nature of multi-agent systems means agents may be designed and developed independently. Any implementation of interaction protocols must therefore support agents with heterogeneous technical requirements (e.g. hardware, platform, or programming language) and diverse implementations.

For this reason, we have created a format for describing interactions and protocols according to the formalism in Section 3.1, while being independent of specific programming languages in which agents may be implemented. The format is based on YAML [10], itself a language for structured data that is both machine and human-readable (and as such, easy to edit with any standard text editor).

An example protocol specification corresponding with the bilateral negotiation example of Section 3.2, as well as technical description of specification's format can be found in Section A in the appendix.

4.3 From Specification to Code

A protocol specification is only a high-level description of an interaction protocol and is designed to be independent of agents' implementations. For agents to engage in an interaction however, they need to have access to the protocol's definitions in the language they use. For instance, an agent developed in Python who wants to negotiate using the protocol in Section 3.2 with another agent developed in Go, each require an interpretation of the same specification in their own language.

The framework thus includes a *protocol generator*, which for any agent, given a protocol specification, produces the protocol package in the language this particular agent uses. Currently, the generator only takes into account the programming language that agents use. This architecture however allows for other agent constraints to be added later for consideration by the generator (e.g. an agent with limited resources may receive a more resource-bounded interpretation of a protocol's data structures).

Any generated protocol package consists of a) the technical definitions of the underlying concepts, e.g. a message, speech-acts and the reply structure, b) verification checks on messages according to the protocol, and c) description of how messages may be serialised and deserialised. Instructions on how to generate a protocol package from a specification is given in Section C in the appendix.

4.4 Serialisation

The messages agents exchange in dialogues may contain arbitrary contents with local representations (i.e. objects). However, in order to send these messages over a network, their local representations must be serialised by the sender and deserialised by the receiver. The framework uses Protocol Buffers [29] as the serialisation mechanism for cross-platform support.

Upon generating a protocol, the generator produces a protocol buffer schema, describing serialised models, as well as encoding and decoding logic corresponding with the protocol's specification. The protocol buffer schema generated for bilateral negotiations specification can be found in Section B of the appendix.

4.5 Protocol Adherence Verification

A protocol package, generated to meet the needs of an agent, provides all the definitions needed for this agent to know "*what can be done in these interactions?*". The question, "*what to do in an interaction?*", naturally arising, must be addressed for agents to partake in and benefit from interactions.

The framework places a separation of concerns with a clear distinction between the roles of a protocol designer and an agent developer. The former designs an interaction protocol whose constructs are publicly accessible (e.g. what is a valid reply to any speech-act). The latter designs the agent, most likely independent of the protocol designer, and has access to constructs privately owned by its agent (e.g. the agent's utility function). The second question is addressed by agent developers, who create strategies for their agents to engage in specific interactions.

The peer-to-peer nature of communication between agents, a direct consequence of the environment's decentralised nature, also means that in any interaction, the participating players' adherence to the protocol is verified only by the players themselves. Any message m is thus verified by its sender before being sent and by its receiver right after it is received. The dialogue-based design of interaction protocols enables the resolution of errors and failures of compliance via dialogues themselves [24].

5 Related Work

One of the most widely used notations for designing interaction protocols is Agent UML (AUML) [26]. AUML is an extension of UML 2.0 [42] with additional agent-specific features. AUML is a graphical notation and for relatively simple interactions is intuitive. However, it is one of the most complex notations [36] with 17 distinct graphical constructs compared with 11 for Statechart [22], and 5 or fewer for other graphical notations [1,41,49].

AUML assumes message delivery is synchronous, therefore protocols suffer from the *enactability problem* [18] which has to be addressed externally. The reliance on UML also presents a major issue for AUML in that it is a semi-formal language. There are no formal semantics for Interaction Diagrams and some of its elements make use of unstructured text (e.g. guards). This means ambiguities and misunderstandings are possible, which in turn makes the realisation of tools and implementations of AUML Interaction Diagrams difficult. Therefore, AUML is not considered a precise language.

Statechart is another popular and highly influential notation for agent interaction protocols [22]. Similar to AUML, Statechart is a graphical notation, but unlike AUML, it supports variables and parallel protocols, allowing them to model information-driven interactions. Also similar to AUML, interaction protocols in statecharts are designed with the synchronicity assumption, which means Statechart protocols are prone to the same enactability problem.

Compared to FSMs and Petri nets, Statechart's notation is fairly complex, both graphically (with 11 distinct graphical elements) and due to unstructured text in certain elements (e.g. guards and effects). Furthermore, there are more than twenty semantics, of differing types proposed for Statecharts. This means the same statechart can be interpreted completely differently under different semantics [17,45].

Statecharts were designed from the outset for reactive event-driven distributed systems, and not multi-agent systems. Therefore, they do not support the notion of roles, and transitions represent events rather than messages. This means that message attributes such as sender/receiver are not specified.

Hierarchical Agent Protocol Notation (HAPN) is a relatively more recent proposal [49]. HAPN focuses on addressing the problems its authors found in prior proposals, namely, a) flexible data-driven protocols, b) role representation and mapping to agents, and c) hierarchical modularity.

HAPN is a graphical notation, with some structured textual elements, and uses Hierarchical Finite State Machines [1] as its underlying conceptual model. It allows modelling parallelism and exceptions, supports information-driven interactions by adding

flexibility on order of messages, and has some support for protocols with multiple role instances.

Similar to the other two notations above, HAPN assumes synchronous message delivery and the authors acknowledge the problems with this assumption, but argue that this is a standard and long-standing assumption in protocol design notations and suggest external processes for addressing them, e.g. [18].

Other research strands less closely related to our proposal include, commitment-based [51,8,48] and norm-based [5,16] interaction protocols, and BSPL [44]. They are all promising proposals, though some have fundamentally different assumptions than ours (e.g. norm-based methods are usually applied in organisational settings which are not entirely decentralised). However, none of these proposals has yet matured into a widespread practical methodology.

6 Conclusion

In this work, we propose a general framework for dialogue-based bilateral interactions that use protocols to govern their participants' behaviours. There are many proposals in the literature that focus on one or some aspects of the above problem.What sets this work apart is its end-to-end approach, from formalisation and specification of interactions to implementation and deployment as part of agent solutions.

The multi-agent and strictly decentralised nature of the environments we focus on (see **Req. 1**) requires that protocols are verifiable by interaction participants and support asynchronous message delivery. The practicality requirement (see **Req. 2**) means the framework has to be formal and precise to be computational, and easy to use to be practical. Although it is straightforward to verify the former via practical use, the latter is harder to measure.

Our end-to-end approach means that we focused on a minimal complete proposal. It can be further improved and extended in various directions, some of which outlined below:

- The replying nature of messages in dialogues lends itself nicely to a graph-theoretical interpretation. This, in turn, would enable studying the properties of interactions under graph-theoretical assumptions. It would also facilitate the creation of tools for visualising and more easily analysing interactions, and further informs a natural definition for protocol modularity (i.e. sub-protocols) via sub-graphs.
- Although the framework is fully formalised and practically accessible, the properties of its interaction instances are left unspecified. It would be useful to present these properties and highlight the effects of different interaction designs, including best practices, on the characteristics of the resulting interactions.
- Another line of work could focus on increasing the expressiveness of the protocol specification language and the cross-platform support of the protocol generator, covering more programming languages and platforms, and meeting other technical requirements by agents, thus increasing support for more heterogeneous agents.

```
---
name: negotiation
author: EMAS_authors
version: 0.1.0
description: 'A protocol for bilateral negotiations.'
license: Apache-2.0
aea_version: '>=1.0.0, <2.0.0'
protocol_specification_id: EMAS_authors/negotiation:1.0.0
speech_acts:
  cfp:
    e: ct:Resources
  propose:
    e: ct:Resources
    p: pt:float
  accept: {}
  decline: {}
...
---
ct:Resources: |
  bytes resources_bytes = 1;
...
---
initiation: [cfp]
reply:
  cfp: [propose, decline]
  propose: [propose, accept, decline]
  accept: []
  decline: []
termination: [accept, decline]
roles: {b, s}
end_states: [agreement_reached, agreement_unreached]
keep_terminal_state_dialogues: true
...
```

Listing 1: Protocol Specification for Bilateral Negotiation

A Protocol Specification

Listing 1 shows the protocol specification corresponding with bilateral negotiation in Section 2.3. Protocol specifications are formatted in YAML.[4] and consist of three YAML documents (enclosed between $---$ and ...):

- The first document contains basic information about the protocol as well as its speech-acts. Speech-acts are each listed as key-values, where the key is the performative and the value is a dictionary of its contents specifying their name and type. For example, *cfp* has one content, named `e` whose type is `ct : Resource`. The specification also comes with a language-independent type system. A summary of the types are in Table 1.
- The second document contains protocol buffer schema snippet of any custom types defined for speech-act contents.
- The third document contains the interaction protocols definitions where the fields are self-explanatory and correspond with definitions in Section 3.1.

B Protocol Buffer Schema

An example of the protocol buffer schema that a generator produces from the specification in Listing 1 is given in Listing 2.

C Instructions on Using the Framework

Note that detailed and up-to-date instructions can be found at https://github.com/fetchai/agents-aea.

- Ensure you have Python 3.7 installed on your machine.
- Install the AEA framework using pip (python package installer):
  ```
  > pip install aea[all]
  ```
- You many need to create a registry account (this is so you can publish your agent's packages on a registry):
  ```
  > aea init
  ```
  Then follow the on-screen instructions.
- Create an agent:
  ```
  > aea create agent
  ```
- Enter the newly created agent directory:
  ```
  > cd agent
  ```
- Generate the protocol:
  ```
  > aea generate protocol <path>
  ```
  where ⟨`path`⟩ is the path to the protocol specification file.
- The protocol package can now be found under . . . `/agent/protocols/negotiation`.

[4] See https://yaml.org

Code	Type	Format	Example	In Python
⟨CT⟩	Custom types	ct:RegExp(^[A − Z][a − zA − Z0 − 9] ∗ $)	ct:DataModel	Custom Class
⟨PT⟩	Primitive types	pt:bytes	pt:bytes	bytes
		pt:int	pt:int	int
		pt:float	pt:float	float
		pt:bool	pt:bool	bool
		pt:str	pt:str	str
⟨PCT⟩	Primitive collection types	pt:set[⟨PT⟩]	pt:set[pt:str]	FrozenSet[str]
		pt:list[⟨PT⟩]	pt:list[pt:int]	Tuple[int, . . .]
⟨PMT⟩	Primitive mapping types	pt:dict[⟨PT⟩, ⟨PT⟩]	pt:dict[pt:str, pt:bool]	Dict[str, bool]
⟨MT⟩	Multi types	pt:union[⟨PT⟩/⟨CT⟩/⟨PCT⟩/⟨PMT⟩, . . . , ⟨PT⟩/⟨CT⟩/⟨PCT⟩/⟨PMT⟩]	pt:union[ct:Model, pt:set[pt:str]]	Union[Model, FrozenSet[str]]
⟨O⟩	Optional	pt:optional[⟨MT⟩/⟨PT⟩/⟨CT⟩/⟨PCT⟩/⟨PMT⟩]	pt:optional[pt:bool]	Optional[bool]

Table 1: Protocol specification content types

```
syntax = "proto3";

package aea.EMAS_authors.negotiation;

message NegotiationMessage{

    // Custom Types
    message Resources{
        bytes resources_bytes = 1;
    }


    // Performatives and contents
    message Cfp_Performative{
        Resources e = 1;
    }

    message Propose_Performative{
        Resources e = 1;
        float p = 2;
    }

    message Accept_Performative{}

    message Decline_Performative{}


    oneof performative{
        Accept_Performative accept = 5;
        Cfp_Performative cfp = 6;
        Decline_Performative decline = 7;
        Propose_Performative propose = 8;
    }
}
```

Listing 2: Protocol Specification for Bilateral Negotiation

References

1. Alur, R., Kannan, S., Yannakakis, M.: Communicating hierarchical state machines. In: Wiedermann, J., van Emde Boas, P., Nielsen, M. (eds.) Automata, Languages and Programming. pp. 169–178. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
2. Amgoud, L., Dimopoulos, Y., Moraitis, P.: A unified and general framework for argumentation-based negotiation. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems. pp. 158:1–158:8. AAMAS '07, ACM, New York, NY, USA (2007)

3. Amgoud, L., Parsons, S., Maudet, N.: Arguments, dialogue, and negotiation. In: ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence, Berlin, Germany, August 20-25, 2000. pp. 338–342 (2000)
4. Amgoud, L., Vesic, S.: A formal analysis of the role of argumentation in negotiation dialogues. Journal of Logic and Computation **22**(5), 957–978 (2012)
5. Andrighetto, G., Governatori, G., Noriega, P., van der Torre, L.: Normative Multi-Agent Systems. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (04 2013)
6. Arcos, J.L., Esteva, M., Noriega, P., Rodríguez-Aguilar, J.A., Sierra, C.: Engineering open environments with electronic institutions. Engineering Applications of Artificial Intelligence **18**(2), 191 – 204 (2005), agent-oriented Software Development
7. Austin, J., Austin, J., Urmson, J., Urmson, J., Sbisà, M.: How to Do Things with Words. A Harvard paperback, Harvard University Press (1975)
8. Baldoni, M., Baroglio, C., Marengo, E., Patti, V.: Constitutive and regulative specifications of commitment protocols: A decoupled approach. ACM Trans. Intell. Syst. Technol. **4**(2) (Apr 2013)
9. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology). John Wiley & Sons, Inc., Hoboken, NJ, USA (2007)
10. Ben-Kiki, O., Evans, C., Ingerson, B.: Yaml ain't markup language (YAML™) version 1.2. Tech. rep., YAML (2009)
11. Black, E., Hunter, A.: An inquiry dialogue system. Autonomous Agents and Multi-Agent Systems **19**(2), 173–209 (2009)
12. Boissier, O., Bordini, R.H., Hübner, J.F., Ricci, A., Santi, A.: Multi-agent oriented programming with jacamo. Science of Computer Programming **78**(6), 747 – 761 (2013)
13. Castagna, G., Dezani-Ciancaglini, M., Padovani, L.: On global types and multi-party sessions. In: Bruni, R., Dingel, J. (eds.) Formal Techniques for Distributed Systems. pp. 1–28. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
14. Chopra, A., V, S., Singh, M.: An evaluation of communication protocol languages for engineering multiagent systems. Journal of Artificial Intelligence Research **69**, 351–1393 (Dec 2020). https://doi.org/10.1613/jair.1.12212
15. Collier, R.W., Russell, S., Lillis, D.: Exploring aop from an oop perspective. In: Proceedings of the 5th International Workshop on Programming Based on Actors, Agents, and Decentralized Control. p. 25–36. AGERE! 2015, Association for Computing Machinery, New York, NY, USA (2015)
16. Dastani, M., van der Torre, L., Yorke-Smith, N.: Commitments and interaction norms in organisations. Autonomous Agents and Multi-Agent Systems **31**, 207–249 (2015)
17. Eshuis, R.: Reconciling statechart semantics. Science of Computer Programming **74**(3), 65 – 99 (2009)
18. Ferrando, A., Winikoff, M., Cranefield, S., Dignum, F., Mascardi, V.: On enactability of agent interaction protocols: Towards a unified approach. In: Dennis, L.A., Bordini, R.H., Lespérance, Y. (eds.) Engineering Multi-Agent Systems. pp. 43–64. Springer International Publishing, Cham (2020)
19. Freire, J., Botelho, L.: Executing explicitly represented protocols. In: In Workshop on challenges in open systems at AAMAS'02 (2002)
20. Gregori, M., Palanca, J., Aranda, G.: A jabber-based multi-agent system platform. In: Proceedings of the International Conference on Autonomous Agents. vol. 2006, pp. 1282–1284 (2006)
21. Hanachi, C., Sibertin-Blanc, C.: Protocol moderators as active middle-agents in multi-agent systems. Autonomous Agents and Multi-Agent Systems **8**, 131–164 (2004)
22. Harel, D.: Statecharts: a visual formalism for complex systems. Science of Computer Programming **8**(3), 231 – 274 (1987)

23. Herlihy, M., Rajsbaum, S., Tuttle, M.R.: Unifying synchronous and asynchronous message-passing models. In: Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing. p. 133–142. PODC '98, Association for Computing Machinery, New York, NY, USA (1998)
24. Hosseini, S.A.: Dialogues Incorporating Enthymemes and Modelling of Other Agents' Beliefs. Ph.D. thesis, King's College London (September 2017)
25. Huget, M.P., Odell, J.: Representing agent interaction protocols with agent uml. In: Odell, J., Giorgini, P., Müller, J.P. (eds.) Agent-Oriented Software Engineering V. pp. 16–30. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
26. Huget, M.P., Odell, J., Bauer, B.: The AUML Approach, pp. 237–257. Springer US, Boston, MA (2004)
27. Hulstijn, J.: Dialogue Models For Inquiry and Transaction. Ph.D. thesis, Universiteit Twente, Proefschrift Universiteit Twente, The Netherlands (2000)
28. Kakas, A., Maudet, N., Pavlos, M.: Modular representation of agent interaction rules through argumentation. Autonomous Agents and Multi-Agent Systems **11** (09 2005). https://doi.org/10.1007/s10458-005-2176-4
29. Kaur, G., Fuad, M.M.: An evaluation of protocol buffer. In: Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon). pp. 459–462 (2010)
30. Leppänen, T., Lacasia, J.A., Tobe, Y., Sezaki, K., Riekki, J.: Mobile crowdsensing with mobile agents. Autonomous Agents and Multi-Agent Systems **31**, 1–35 (2015)
31. McCorry, P., Buckland, C., Bakshi, S., Wüst, K., Miller, A.: You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies. In: Bracciali, A., Clark, J., Pintore, F., Rønne, P.B., Sala, M. (eds.) Financial Cryptography and Data Security. pp. 35–49. Springer International Publishing, Cham (2020)
32. Minarsch, D., Hosseini, S.A., Favorito, M., Ward, J.: Autonomous economic agents as a second layer technology for blockchains: Framework introduction and use-case demonstration. In: 2020 Crypto Valley Conference on Blockchain Technology (CVCBT). pp. 27–35 (2020)
33. Minarsch, D., Favorito, M., Hosseini, A., Ward, J.: Trading agent competition with autonomous economic agents. In: Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems. p. 2107–2110. AAMAS '20, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2020)
34. Minarsch, D., Favorito, M., Hosseini, S.A., Turchenkov, Y., Ward, J.: Autonomous economic agent framework. In: Engineering Multi-Agent Systems. (to publish) (2021)
35. Minarsch., D., Hosseini., S.A., Favorito., M., Ward., J.: Trading agent competition with autonomous economic agents. In: Proceedings of the 13th International Conference on Agents and Artificial Intelligence - Volume 1: SDMIS,. pp. 574–582. INSTICC, SciTePress (2021). https://doi.org/10.5220/0010431805740582
36. Moody, D., van Hillegersberg, J.: Evaluating the visual syntax of uml: An analysis of the cognitive effectiveness of the uml family of diagrams. In: Gašević, D., Lämmel, R., Van Wyk, E. (eds.) Software Language Engineering. pp. 16–34. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
37. Parsons, S., Wooldridge, M., Amgoud, L.: Properties and complexity of some formal inter-agent dialogues. Journal of Logic and Computation **13**(3), 347–376 (2003)
38. Prakken, H.: Coherence and flexibility in dialogue games for argumentation. Journal of Logic and Computation **15**(6), 1009–1040 (2005)
39. Prakken, H.: Formal systems for persuasion dialogue. The Knowledge Engineering Review **21**, 163–188 (5 2006)
40. Rahmani, L., Minarsch, D., Ward, J.: Peer-to-peer autonomous agent communication network. In: Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems. p. [to appear]. AAMAS '21, International Foundation for Autonomous Agents and Multiagent Systems (2021)

41. Reisig, W.: Petri Nets: An Introduction. Springer-Verlag, Berlin, Heidelberg (1985)
42. Rumbaugh, J., Jacobson, I., Booch, G.: Unified Modeling Language Reference Manual, The (2nd Edition). Pearson Higher Education (2004)
43. Sadri, F., Toni, F., Torroni, P.: Logic agents, dialogues and negotiation: an abductive approach. In: In Proceedings of AISB'01 Convention. pp. 71–78. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (2001)
44. Singh, M.P.: Information-driven interaction-oriented programming: Bspl, the blindingly simple protocol language. In: The 10th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. p. 491–498. AAMAS '11, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2011)
45. Taleghani, A., Atlee, J.M.: Semantic variations among uml statemachines. In: Nierstrasz, O., Whittle, J., Harel, D., Reggio, G. (eds.) Model Driven Engineering Languages and Systems. pp. 245–259. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
46. Walton, D.N., Krabbe, E.C.: Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning. State University of New York, State University of New York Press (1995)
47. Wattenhofer, R.: Distributed Ledger Technology: The Science of the Blockchain. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2nd edn. (2017)
48. Winikoff, M., Liu, W., Harland, J.: Enhancing commitment machines. In: Leite, J., Omicini, A., Torroni, P., Yolum, p. (eds.) Declarative Agent Languages and Technologies II. pp. 198–220. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
49. Winikoff, M., Yadav, N., Padgham, L.: A new hierarchical agent protocol notation. Autonomous Agents and Multi-Agent Systems **32**, 59—-133 (07 2018)
50. Wooldridge, M.: An Introduction to MultiAgent Systems. Wiley, 2 edn. (June 2009)
51. Yolum, P., Singh, M.P.: Commitment machines. In: Meyer, J.J.C., Tambe, M. (eds.) Intelligent Agents VIII. pp. 235–247. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
52. Yoshida, N., Hu, R., Neykova, R., Ng, N.: The scribble protocol language. In: Abadi, M., Lluch Lafuente, A. (eds.) Trustworthy Global Computing. pp. 22–41. Springer International Publishing, Cham (2014)