

PROTOCOL SYNTHESIS WITH DIALOGUE STRUCTURE THEORY

Jarred McGinnis

David Robertson

Chris Walton

*Centre for Intelligent Systems and Applications
University of Edinburgh
11 Crichton Street
Edinburgh, Scotland EH8 9LE*

Abstract

Inspired by computational linguistic approaches to annotate the structures that occur in human dialogue, this paper describes a technique which encodes these structures as transformations applied to a protocol language. Agents can have a controlled and verifiable mechanism to synthesise and communicate their interaction protocol during their participation in a multiagent system. This is in contrast to the approaches where agents must subscribe to a fixed protocol and relinquish control over an interaction that may not satisfy the agent's dialogical needs or rely on its internal reasoning to determine which message to communicate at a certain point in the dialogue.

1 Introduction

Research into agent communication is producing increasingly more robust models. Much of this research has turned to other disciplines for inspiration. Philosophy and Linguistics have a several thousand year head start in reflecting upon the nature of communication [2]. These thinkers are concerned with human communication in particular, but insights and models they have developed are readily applicable to the study of agent communication. BDI-logics [7], speech acts [6], social commitment [20] and argumentation [1] have originated in the works of philosophers and linguists [4, 19, 21].

Many have been attracted to a societal view of communication. They take the position that communicating entities, whether they be organic or synthesised, do not communicate in a vacuum but rather in the context of the society made of the other communicating entities around it. This society has rules which govern the behaviour of the agents, constraining the members to perform in accordance with a set of implicit or explicit protocols. Participants in the society willingly sacrifice autonomy and submit to these protocols in order to gain a measure of utility or to accomplish a goal of more value than the independence lost. Traditionally, protocols have been seen as static and inflexible and usually defined as specifications for a human engineer to interpret and encode his or her agent with that interpretation. The approach described in this paper addresses the possibility of a protocolled approach to communication where the agents themselves not only communicate the protocol to each other but also create the protocol during the interaction.

Protocols are not only created with respect to societal conventions but the act of communication itself has conventions to which speakers adhere. Linguists have been interested in this phenomena. They have also adopted this study for the purposes of annotating human dialogue to aid its study for the purposes of automated text analysis and generation [5]. The challenges this field faces largely differ from the concerns of multiagent communication, such as anaphoric ambiguities, but there are findings which add a robustness to the protocolled societal approach to agent communication.

Relationships exist between messages regardless of the particular domain with which the messages are concerned. A question implies the anticipation of the eventual occurrence of an answer even if the reply is a shrug of the shoulders. This is regardless of whether that answer be to the question of "What time is it?" or "Can you compare and contrast the post-modern interpretations of abstract expressionism to a random sequence of adjectives?" It is these generalised patterns which exist in human communication that we have adopted for our purposes. The result is the creation of a means to synthesise a protocol which can reproduce the reliable communication produced by other protocolled approaches without being fixed to a static protocol.

We will begin our discussion with an introduction and explanation of dialogue structures in general and within the context of multiagent communication. Section 3 explains the essentials to the protocol language used to implement this approach. The library of transformations which enable the synthesis of protocols is described in section 4, and the process of synthesis is explained in section 5. We conclude in section 6.

2 Using Dialogue Structures

In human dialogue the utterances that the participants make do not occur in isolation. Humans rely on tacit patterns to ground communication. Some have proposed this is the following of certain rules, and others have argued these rules are only descriptions of the process of having a conversation [19]. Regardless, these patterns can be generalised without concern to the content of the messages. The idea for this approach was largely inspired by the works of [5, 3, 18], and the standardisation efforts of Dialogue Structure Theory (DST) for the annotation of human dialogue transcriptions.

There are a number of approaches that more or less could be used for the run time synthesis of interaction protocols. Although each have proved their worth for a variety of multiagent applications, each fails in some aspect to provide the unique advantages found by the use of dialogue structures.

Performatives are a common approach for agent communication, and it may be possible to pack pan-dialogical concerns into individual performatives. Yet, this would be an ungainly implementation and an abuse of the spirit of performatives. They are meant to reflect the conditions and effects of a single communicative act rather than the relationships between them or their place within a sequence of message exchanges. Our concern is more generic than particular performatives in a given ACL. It is our goal to capture the generic structure of conversation that occurs in dialogues regardless of the performative or ACL used.

Planning research has been brought to bear on the problem [12]. Agents use planning techniques to produce an interaction protocol to reach a previously defined goal by means of communicating with other agents. Firstly, much of communication is not driven by clearly defined goals. It could be the goal of the dialogue to determine the goal of the interaction. Planning is also presented with the unique challenges of the agency model. Besides planning's reputation for a paucity in terms of data structures, there is also another difficulty in using planners for this purpose. It will be difficult for a planner to produce anything more robust than a look-ahead planner, because of the unpredictability of other agents. The planning agent would constantly be replanning in reaction to others actions. It would result in a lot of computation without much satisfaction. Even with the help of making assumptions about other agent's rational behaviour, existing approaches still have speed issues for real-time systems. It is for this reason that it would be much more appropriate to have a small set of transformations which the agent can apply mechanically to achieve the same goal. This is exactly what we describe in this paper.

Machine learning is also being applied to the agency paradigm [17]. The techniques of machine learning introduce a number of unnecessary difficulties. For example, it would be helpful to have transparency and readability of the protocols used by the agents to facilitate human/computer interaction or even humans to understand the protocols used which will assist in the design of new agents. Also, the common problem of producing corpora that hounds machine learning for agency is also a problem in synthesising interaction protocols. Similar to planning approaches, the same goal can be achieved with a set of transformations which can free the agent to spend its computation on learning a strategy for the domain rather than the discussion of that domain.

It is correct to point out the work using social commitments, norms, dialogue games, and other such models of communication provides agents with the ability to reason about communication. Yet, we still retain advantages. It is not the goal to replace any particular model of agency. The goal is to exploit the unique advantages provided by the LCC language and framework, but to enhance its flexibility. The transformations are purely dialogical in the sense they are generic operations which unfold a single message protocol to a two message protocol which in turn can be used to synthesise a three message protocol, and so on. The agent receiving the synthesised protocol can follow it blindly without needing to understand that its dialogical actions satisfy some commitment, norm, or rule of a dialogue game. The advantage being touted is a clean and simple dialogically oriented means to drive protocolled communication while maintaining an agent's ability to unilaterally explore dialogical options not currently present in a given protocol. The other unique advantage is that not only can an agent generate its expected moves given its model (e.g. norms, commitments, etc.) but it can also communicate its expectations for others. Whereas, these traditional agent-centric models typically only provide guidance for a single agent and they assume agents share the same model of communication in order to coordinate their conversation.

The details of dialogue structure theory is largely concerned with issues unique to human communication. Our focus on agent communication neatly avoids the most difficult issues associated with this research. DST has been useful for developing metaphors for the development of protocols and protocol synthesis, but its use is superficial. DST, whether used for annotating human dialogue or generating natural language, must concern itself with the minutia and subtleties that software communication does not. All aspects of agent communication is engineered. As a result, there is a regularity, simplicity, and explicitness to it. This artifactual form of communication is not complicated by thousands of years of culture and tradition that complicates human discourse [13]. Having been saved from the most onerous tasks of DST, we are freed to concentrate on the much more modest task at hand which is using some basic ideas from the field to drive protocol synthesis.

3 The Protocol Language

\mathcal{P}	\in	Protocol	$::=$	$\langle S, A^{\{n\}}, K \rangle$	
A	\in	Agent Clause	$::=$	$\theta :: op.$	
θ	\in	Agent Definition	$::=$	agent (R, Id)	
op	\in	Operation	$::=$	no op	
				θ	
				(op)	(Precedence)
				$M \Rightarrow \theta$	(Send)
				$M \Leftarrow \theta$	(Receive)
				$op1$ then $op2$	(Sequence)
				$op1$ or $op2$	(Choice)
				$M \Rightarrow \theta \leftarrow \psi$	(Prerequisite)
				$\psi \leftarrow M \Leftarrow \theta$	(Consequence)
ρ	\in	message	$::=$	$\langle m, \mathcal{P} \rangle$	

Figure 1: An Abstract Syntax of the Protocol Language

Figure 1 defines the syntax of a protocol language taken from [16] which also gives a fuller explanation of the language and framework. The protocol consists a set of agent clauses, $A^{\{n\}}$. These clauses are defined by an agent definition made up of a role (R) and unique identification (Id). A role is defined in a similar way as Electronic Institutions: It is a way of defining communicative activity for a group of agents rather than individuals. This agent definition is expanded by a number of operations.

Operations can be classified in three ways: actions, control flow, and conditionals. Actions are the sending or receiving of messages, a no op, or the adoption of a role. Control Flow operations temporally order the individual actions. Actions can be put in sequence (one action must occur before the other), or given a choice point (one and only one action should occur before any further action). The ' \Rightarrow ' and ' \Leftarrow ' denote messages, M , being sent and received. On the left-hand side of the double arrow is the message and on the right-hand side is the other agent involved in the interaction.

Constraints can fortify or clarify semantics of the protocols. Those occurring on the left of the ' \Leftarrow ' are postconditions and those occurring on the right are preconditions. The symbol ψ represents a first order propositions. For example, an agent receiving a protocol with the constraint to believe a proposition s upon being informed of s can infer that the agent sending the protocol has a particular semantic interpretation of the act of informing other agents of propositions.

A message is defined as the tuple, $\langle m, \mathcal{P} \rangle$. Where m is the message an agent is currently communicating, and \mathcal{P} is the protocol written using the language described in figure 1. The protocol, in turn, is a triple, $\langle S, A^{\{n\}}, K \rangle$. S is the dialogue state. This is a record of the path of the dialogue through the conversation space and the current state of the dialogue for the agents. This set of agent clauses is marked to show the progress of the dialogue and the current state of the interaction. The messages are marked as closed or failed depending on whether they are communicated successfully. Messages which have been communicated are encased by a 'c', $c(M)$.

A Operation is decided to be closed, meaning that it has been covered by the preceding interaction, as

follows:

$$\begin{aligned}
& \text{closed}(c(X)) \\
& \text{closed}(A \text{ or } B) \leftarrow \text{closed}(A) \vee \text{closed}(B) \\
& \text{closed}(A \text{ then } B) \leftarrow \text{closed}(A) \wedge \text{closed}(B) \\
& \text{closed}(X ::= D) \leftarrow \text{closed}(D)
\end{aligned}$$

$\text{satisfied}(C)$ is true if C can be solved from the agent's current state of knowledge. $\text{satisfy}(C)$ is true if the agent's state of knowledge can be made such that C is satisfied. The bottom of figure 2 should read that $\text{clause}(\mathcal{P}, X)$ is true if clause X appears in the dialogue framework of protocol \mathcal{P} , as defined in figure 1.

The second part is a set of agent clauses, $A^{\{n\}}$, necessary for the dialogue. The protocol also includes a set of axioms, K , consisting of common knowledge to be publicly known between the participants. This explicit communication of the dialogue state provides a means of coordination. It is possible to create an agent which retains no internal record of the state of the dialogue but rather uses the communicated dialogue state as a book mark for which to hold its place and remind it of the next communicative step it can take.

Agents themselves communicate the conventions of the dialogue. This is accomplished by the participating agents satisfying two simple engineering requirements. Agents are required to share a dialogical framework. This an unavoidable necessity in any meaningful agent communication. This includes the requirements on the individual messages are expressed in a ontology understood by the agents. The issue of ontology mapping is still open, and its discussion extends beyond the scope of this paper. The second requirement obligates the agent to provide a means to interpret the received message and its protocol. The agent must be able to unpack a received protocol, find the appropriate actions it may take, and update the dialogue state to reflect any actions it chooses to perform.

$$\begin{aligned}
& A ::= B \xrightarrow{M_i, M_o, \mathcal{P}, O} A ::= E \\
& \text{if } B \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ or } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& \text{if } \neg \text{closed}(A_2) \wedge A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ or } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& \text{if } \neg \text{closed}(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ then } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \text{ then } A_2 \\
& \text{if } A_1 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& A_1 \text{ then } A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} A_1 \text{ then } E \\
& \text{if } \text{closed}(A_1) \wedge A_2 \xrightarrow{M_i, M_o, \mathcal{P}, O} E \\
& C \leftarrow M \leftarrow A \xrightarrow{M_i, M_i - \{M \leftarrow A\}, \mathcal{P}, \emptyset} c(M \leftarrow A) \\
& \text{if } (M \leftarrow A) \in M_i \wedge \text{satisfy}(C) \\
& M \Rightarrow A \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \{M \Rightarrow A\}} c(M \Rightarrow A) \\
& \text{if } \text{satisfied}(C) \\
& \text{null} \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \emptyset} c(\text{null}) \\
& \text{if } \text{satisfied}(C) \\
& \text{agent}(r, id) \leftarrow C \xrightarrow{M_i, M_o, \mathcal{P}, \emptyset} a(R, I) ::= B \\
& \text{if } \text{clause}(\mathcal{P}, a(R, I) ::= B) \wedge \text{satisfied}(C)
\end{aligned}$$

Figure 2: Rules for expanding an agent clause

Figure 2 describes rules for expanding the received protocols. Details can be found in [14]. A similar language for web services is described in [15]. An agent receives a message of the form specified in figure 1. The message is added to the set of messages, M_i , currently being considered by the agent. The agent takes the clause, C_i , from the set of agent clauses received as part of \mathcal{P} . This clause provides the agent with its role in the dialogue. The agent then expands C_i by the application of the rules in figure 2. The expansion is done with respect to the different operators encountered in the protocol and the response to M_i . The result is a new dialogue state, C_n ; a set of output messages, O_n and a subset of M_i , which is the remaining messages

to be considered, M_n . The result is arrived at by applying the rewrite rules. The sequence would be similar to figure 3. C_n is then sent as part of \mathcal{P} which will accompany the sending of each message in O_n .

$$\langle C_i \xrightarrow{M_i, M_{i+1}, \mathcal{P}, O_i} C_{i+1}, \dots, C_{n-1} \xrightarrow{M_{n-1}, M_n, \mathcal{P}, O_n} C_n \rangle$$

Figure 3: Sequence of rewrites

This protocol language is well suited for our purposes. By distributing the protocol during the interaction, the agents have providence over the interaction protocol allowing agents to make transformations. The explicit transmission of the dialogue state records and communicates the choices made as the protocol is realised.

It also catalogues the transformations made and the resulting properties which now hold because of those changes. This allows the mechanism for the flexible protocolled approach we seek. Now, that transformations are possible it is important to ensure they are controlled and meaningful.

4 Transformations

There are various structures which occur in human dialogue which have a different semantic interpretation but share the same syntactical shape. For example, a question followed by an answer has the same structure as a statement and a confirmation. An agent sends a message which is followed by another message being received. It is therefore useful to generalise the vocabulary of transformations to those whose semantics can be uniquely identified by its syntactic structure. Otherwise a kind of semantic leakage occurs and ambiguity seeps into the dialogue and protocol. The sort of distinctions of a question and answer versus a propose and accept should be dealt with at the ACL level. Our concern is makes no assumptions about the particular locutions used for the protocol.

In the figure 4 we define a set of transformations. The number is restricted to all the valid syntactical amendments to the simplest protocol (the protocol of a single message being communicated). The process of pruning away errant transformations are shown in the unfortunately monstrous figure 5.

The library of transformations in figure 4 was created by taking all the possible permutations of the two message protocol given an atomic protocol- an atomic protocol being defined as a single message being communicated, as a more simpler (non-empty) protocol can not be conceived. Each of these single message protocols can be expanded to a two message protocol by the addition of an ‘ **then** ’ or an ‘ **or** ’ operator followed by another message either incoming or outgoing. The total number of these two message protocols is seventy-two. By excluding protocols not possible within the LCC framework, the set is thinned to twenty-four possible transformations. For example a protocol cannot exist that has a closed message following an open one (e.g. $M1 \leftarrow \theta$ **then** $c(M2 \Rightarrow \theta)$). This is because of the way the protocol is expanded by the LCC framework. The initial set is shown at the top of figure 5.

After the first pruning, the set of twenty-four sheds six:

$$\begin{aligned} M1 \Rightarrow \theta &\longrightarrow M1 \Rightarrow \theta \text{ or } M2 \Leftarrow \theta \\ M1 \Leftarrow \theta &\longrightarrow M1 \Leftarrow \theta \text{ or } M2 \Rightarrow \theta \\ f(M1 \Rightarrow \theta) &\longrightarrow f(M1 \Rightarrow \theta) \text{ or } M2 \Leftarrow \theta \\ c(M1 \Rightarrow \theta) &\longrightarrow c(M1 \Rightarrow \theta) \text{ or } M2 \Leftarrow \theta \\ f(M1 \Leftarrow \theta) &\longrightarrow f(M1 \Leftarrow \theta) \text{ or } M2 \Rightarrow \theta \\ c(M1 \Leftarrow \theta) &\longrightarrow c(M1 \Leftarrow \theta) \text{ or } M2 \Rightarrow \theta \end{aligned}$$

Although not strictly illegal there will never be the ambiguity of whose turn it is to speak. This is due to our limiting protocol synthesis to dialogues. Protocols for multiparty conversations could indeed have this ambiguity.

Failure is defined as both the inability to communicate at the semantic level (i.e. the message was sent and received but not sensible with respect to an agent’s knowledge base) as well as the physical failure to send or receive a message. Either a message being received or a message being sent is considered failed. When the failed message is outgoing, the sending agent has marked the failure in the dialogue state (i.e. That agent knows about the failure). The only possible transformation which should be applied is the sending of a second message, a *correction*. Another four transformations being cast away by this second pruning.

Before a Message is Sent		
$M1 \Rightarrow \theta$	$\xrightarrow{\text{response}(M1,M2)}$	$M1 \Rightarrow \theta$ then $M2 \Leftarrow \theta$
$M1 \Rightarrow \theta$	$\xrightarrow{\text{continuation}(M1,M2)}$	$M1 \Rightarrow \theta$ then $M2 \Rightarrow \theta$
$M1 \Rightarrow \theta$	$\xrightarrow{\text{counter}(M1,M2)}$	$M1 \Rightarrow \theta$ or $M2 \Rightarrow \theta$
Before a Message is Received		
$M1 \Leftarrow \theta$	$\xrightarrow{\text{continuation}(M1,M2)}$	$M1 \Leftarrow \theta$ then $M2 \Leftarrow \theta$
$M1 \Leftarrow \theta$	$\xrightarrow{\text{response}(M1,M2)}$	$M1 \Leftarrow \theta$ then $M2 \Rightarrow \theta$
$M1 \Leftarrow \theta$	$\xrightarrow{\text{counter}(M1,M2)}$	$M1 \Leftarrow \theta$ or $M2 \Leftarrow \theta$
Upon the Reception of a Message		
$c(M1 \Leftarrow \theta)$	$\xrightarrow{\text{clarification}(M1,M2)}$	$c(M1 \Leftarrow \theta)$ then $M2 \Rightarrow \theta$
Upon Failure of a Message		
$f(M1 \Rightarrow \theta)$	$\xrightarrow{\text{correction}(M1,M2)}$	$f(M1 \Rightarrow \theta)$ then $M2 \Rightarrow \theta$
$f(M1 \Leftarrow \theta)$	$\xrightarrow{\text{correction}(M1,M2)}$	$f(M1 \Leftarrow \theta)$ then $M2 \Rightarrow \theta$

Figure 4: The Vocabulary of Transformations

$$\begin{aligned}
f(M1 \Rightarrow \theta) &\longrightarrow f(M1 \Rightarrow \theta) \text{ then } M2 \Leftarrow \theta \\
f(M1 \Leftarrow \theta) &\longrightarrow f(M1 \Leftarrow \theta) \text{ then } M2 \Leftarrow \theta \\
f(M1 \Rightarrow \theta) &\longrightarrow f(M1 \Rightarrow \theta) \text{ or } M2 \Rightarrow \theta \\
f(M1 \Leftarrow \theta) &\longrightarrow f(M1 \Leftarrow \theta) \text{ or } M2 \Rightarrow \theta
\end{aligned}$$

It is not possible to make a transformation on the closed atomic protocol of a single sent message. By the definition of LCC, the message has already been communicated and with it the protocol one wishes to transform. For this reason, we can dismiss any protocol synthesised upon a closed outgoing message such as these:

$$\begin{aligned}
c(M1 \Rightarrow \theta) &\longrightarrow c(M1 \Rightarrow \theta) \text{ then } M2 \Leftarrow \theta \\
c(M1 \Rightarrow \theta) &\longrightarrow c(M1 \Rightarrow \theta) \text{ then } M2 \Rightarrow \theta \\
c(M1 \Rightarrow \theta) &\longrightarrow c(M1 \Rightarrow \theta) \text{ or } M2 \Rightarrow \theta
\end{aligned}$$

This point, $c(M1 \Rightarrow \theta)$, in the dialogue state occurs after the agent has evaluated, made its decision with respect to the conversation, and has expanded the protocol and only just before the message with the protocol and the dialogue state are sent to the other agent. It would be too late to synthesise more protocol. The situation is different for an incoming message which has been closed. The agent has just received the message. It has marked the message as closed and is at the point to make a decision.

$$\begin{aligned}
c(M1 \Leftarrow \theta) &\longrightarrow c(M1 \Leftarrow \theta) \text{ then } M2 \Leftarrow \theta \\
c(M1 \Leftarrow \theta) &\longrightarrow c(M1 \Leftarrow \theta) \text{ or } M2 \Leftarrow \theta
\end{aligned}$$

For a closed received message the only transformation which can be applied is the addition of an outgoing message. From this final pruning, five more transformations can be scratched from the list leaving a more manageable nine shown in figure 4. The bottom of Figure 5 shows the exhaustive set of the only possible syntactic transformations from an atomic protocol to one with two steps. Given all the possible two step protocols, one can apply the transformations to each of those and have all the permutations of a three step protocol, and in turn apply the transformations again to have all four step protocols. This can be done indefinitely in order to represent all possible protocols used within the LCC framework. There is no universally acceptable model of conversation, but we can map phenomena identified in DST research to the identified transformations. This is not implying that the transformation is an exact match to all similar phenomena in human dialogue, but that the mapping have an easily identifiable similarity. This is how figure 4 is derived from the final nine in figure 5.

In dialogues, humans cue for response by a number of verbal and non-verbal cues. This is captured by the two transformations in figure 4. A message is sent and at some point later a message is received from the same agent. The messages and their content can be said to be a *response*.

During discussions, humans will provide choice to their dialogical partners when appropriate. This same need exists in agent communication. The *counter* transformation allows agents to introduce this type of step

in dialogues. Here we have a departure from the phenomenon occurring in human dialogue, versus agent interaction protocols. Rarely in human dialogue are the options for response so explicitly stated as in our example. In agent communication it is not only common, but usually necessary.

Another feature of human dialogues is the use of cues to signify they wish to continue their turn in the dialogue. The *continuation* transformation enables software agents to do the same. The protocol coordinates whose turn it is to speak and an agent wishing to communicate more than one locution would not need a signalling phrase usually required for polite human dialogue but instead have a protocol allowing the multiple messages to be communicated.

Clarifications and Corrections are of great interest to those studying dialogue structures [8, 3]. Corrections are usually reactions to failures in the dialogue. We have addressed outright failures such as message loss or complete misunderstanding as criteria for a *correction* transformation. Whereas, *clarifications* occur when a message received is understood but found to be wanting in detail. An agent providing a date but the other agent needs a year for the date as well *clarification* versus an agent communicating a seemingly erroneous date such as the tenth day of the seventeenth month *correction*. The message encapsulated by a ‘c’ before the *clarification* transformation represents in the protocol language that the message has been sent. The ‘f’ encapsulation represents a message failure which is the requirement for an agent making a *correction* transformation.

The transformations described are as generic as the LCC framework. There is no assumption of the rational make up of agents, the ACL involved or the domain ontology. In order for the transformations to make sense for a particular domain, it is necessary to define specific instances of the dialogue structures with respect to the domain being discussed and the locutions being communicated. These instances serve as synthesis rules. The figure 7 shows the rules for an agent to produce the dialogical steps to play an information-seeking dialogue game. They dictate for the agent what is considered the all proper responses, counters, continuations, corrections or clarifications given the ACL and domain of the dialogue.

For example, in a *response* the protocol has two messages, one coming in and one going out, separated by the ‘ **then** ’ operator. The synthesis rules for the agent say just what locution can be used for a *response* transformation.

response(ask(X),tell(X)).

The synthesis rule above says that the proper *response* for an *ask* locution is a *tell* and their content is the same. Given this synthesis rule, if the agent, we’ll call him ‘a’, has a protocol which is just the sending of an ask to agent b, written as

$agent(Proposition, a) ::= ask(Proposition) \Rightarrow agent(-, b).$

then we can synthesise a two step protocol which provides the protocol step to allow agent b to respond.

$agent(Proposition, a) ::= ask(Proposition) \Rightarrow agent(-, b)$
then $tell(Proposition) \Leftarrow agent(-, b).$

We take advantage of the common knowledge mechanism in LCC to communicate the synthesis rules. This provides a public evaluation of the rules that synthesised the protocol and the ability for other agents to employ the rule for synthesising. We now turn to describe the process of synthesis.

5 Synthesising Protocols

The process of synthesis progresses forward upon the last synthesised message. This is to prevent transformations such as figure 6. The two responses are performed with respect the first message, *M1*. This could go on indefinitely as the agent repeatedly applies synthesis rules with respect to *M1*. Similarly in human dialogue it is not possible to unsay what has been said. It is only possible to correct what has been said afterward.

This is avoided by stepping forward to the last step of the protocol synthesised, and evaluating whether there are any synthesis rules to apply for that message. This way the protocol continues to expand but only in one direction, forward. The synthesis engine avoids state explosion and replication by halting further synthesis after the use of a *counter* rule. Until by communication or the agents choosing a single path exists, the engine cannot continue to synthesise the protocol.

$$\begin{array}{ccc}
MI \Rightarrow \theta & \xrightarrow{\text{response}(MI, M2)} & MI \Rightarrow \theta \text{ then} \\
& & M2 \Leftarrow \theta \\
MI \Rightarrow \theta \text{ then} & \xrightarrow{\text{response}(MI, M3)} & MI \Rightarrow \theta \text{ then} \\
M2 \Leftarrow \theta & & M3 \Leftarrow \theta \text{ then} \\
& & M2 \Leftarrow \theta
\end{array}$$

Figure 6: An illegal Transformation

LCC deals with meta-dialogical (e.g. deontic) concerns in a number of ways, one of which is the use of constraints. The use of constraints also deals context-dependent dialogical issues. The use of constraints with the synthesis rules also provide this functionality. For example, a synthesis rule can be written like this:

$$\mathbf{response}(ask(X), tell(X) \leftarrow hasPrivileges(X, \theta)).$$

This could be described as the proper response to an *ask* about ‘X’ is a *tell* about ‘X’ but only if the agent θ has privileges to that information. The synthesis engine puts the constraint in the appropriate agent clause in accordance with the syntactical rules of LCC. By the definition of LCC, the construction of a constraint on the left hand side of the \leftarrow may only exist upon a received message (e.g. $MI \Leftarrow \theta$) and having a constraint on the right hand side is for outgoing messages (e.g. $MI \Rightarrow \theta$). Since this is a case, it is unambiguous for the synthesis engine to place the message and constraint onto the correct agent’s clause.

In practice the agent will have a set of synthesis rules. Figure 7 shows the set of synthesis rules which enables an agent to synthesise a protocol similar to the one defined in [11].

- a) $\mathbf{response}(question(P), assert(P)).$
- b) $\mathbf{counter}(assert(P), assert(not(P))).$
- c) $\mathbf{counter}(assert(not(P)), assert(unknown)).$
- d) $\mathbf{response}(assert(R), accept(R)).$
- e) $\mathbf{counter}(accept(R), challenge(R)).$
- f) $\mathbf{response}(challenge(R), assert(S) \leftarrow support(R, S)).$
- g) $\mathbf{response}(assert(S), accept(R) \leftarrow memberOf(R, S)).$
- h) $\mathbf{counter}(accept(R), challenge(R) \leftarrow memberOf(R, S)).$
- i) $\mathbf{response}(accept(P), ack(R)).$
- j) $\mathbf{response}(ack(P), accept(R)).$

Figure 7: Synthesis Rules for an Information Seeking Game

This set of rule consists of several *responses* and *counters*. The last step of the synthesised protocol and the content of the locution informs which rules can be applied. Additionally, further constraints can be defined on synthesis rules to restrict their application. This is subtly different than constraining the occurrence of a message in a synthesised protocol which is shown happening in rules *f*, *g*, and *h*.

6 Conclusions

The use of LCC and the framework provides agents with the ability to communicate their interaction protocols as well as coordinate their own dialogues. Once agents are given this control over their interactions and the social norms as defined by the protocol the possibility of modifying the interaction protocol to address run time needs can be explored. Initial work on this idea was reported in [9, 10]. These papers described the process of making transformations to existing protocols by inserting and deleting portions of protocol. Though this approach worked there was a problem with traceability of the transformations. There was no simple way to identify where and when transformations occurred once the dialogue had ended. By the use of the synthesis rules, the user or agent can trace the construction of the protocol (i.e. given a set of synthesis rules one can construct a given protocol and vice versa).

With the exhaustive set of syntactic transformations, an agent may synthesise any dialogue protocol that can be defined in LCC including the use of constraints. When these transformations are defined by a set of synthesis rules using domain specific knowledge, the agent can synthesise a ‘just in time’ protocol to dynamically explore the conversation space. The protocol is constructed given the present dialogue state rather than the use of a static protocol which had been defined a priori. This is desirable when it is impossible or ungainly to define a protocol beforehand to address all possible paths through the conversation space.

It is recognised that the use of a distributed protocol and allowing agents to modify it during run time presents unique challenges. There are issues such as trust, consensus, writing privileges, etcetera. It is also recognised that the use of protocol synthesis would not be practical for domains with a high degree of uniformity and regularity in its communication. Plain LCC would be a better choice. Although, synthesis could be used for automatic protocol construction. The agent could initially use synthesis but later employ the constructed protocol rather than repeatedly synthesising the same protocol. We have not fully begun to explore this. Currently, its use has been restricted to systems where the dialogue is driven by the messages which occur at execution and allowing the agent to react by applying the appropriate synthesis rule to construct more protocol steps. This is an advantage to traditional agent-centric communication where the agent reasons about which message is should be sent. By exploiting the distribute protocol framework, it constrains the number of communicative actions which can be preformed for itself and its dialogical partners.

References

- [1] Leila Amgoud, Nicolas Maudet, and Simon Parsons. Modeling dialogues using argumentation. In *Proceedings of the Fourth International Conference on MultiAgent Systems (ICMAS-2000)*, page 31. IEEE Computer Society, 2000.
- [2] Aristotle. *Topics*. Clarendon Press, 1997.
- [3] Nicholas Asher and Alex Gillies. Common ground, corrections and coordination. *Argumentation*, 17(4):481–512, 2003.
- [4] Michael Bratman. *Intention, Plans, and Practical Reason*. Havard University Press, 1987.
- [5] Mark G. Core and James F. Allen. Coding dialogues with the DAMSL annotation scheme. In David Traum, editor, *Working Notes: AAI Fall Symposium on Communicative Action in Humans and Machines*, pages 28–35, Menlo Park, California, 1997. American Association for Artificial Intelligence.
- [6] T. Finin, R. Fritzon, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In N. Adam, B. Bhargava, and Y. Yesha, editors, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM'94)*, pages 456–463, Gaithersburg, MD, USA, 1994. ACM Press.
- [7] Mike Georgeff, Barney Pell, Martha Pollack, Milind Tambe, and Mike Wooldridge. The belief-desire-intention model of agency. In Jörg Müller, Munindar P. Singh, and Anand S. Rao, editors, *Proceedings of the 5th International Workshop on Intelligent Agents V : Agent Theories, Architectures, and Languages (ATAL-98)*, volume 1555, pages 1–10. Springer-Verlag: Heidelberg, Germany, 1999.
- [8] Jonathan Ginzburg. Dynamics and the semantics of dialogue. In J. Seligman and D. Westerståhl, editors, *Logic, Language, and Computation*, pages 221–237. CSLI, Stanford, Ca, 1996.
- [9] Jarred McGinnis and David Robertson. Dynamic and distributed interaction protocols. In *Proceedings of the AISB 2004 Convention*, pages 45–54, 2004.
- [10] Jarred McGinnis and David Robertson. Realizing agent dialogues with distributed protocols. In *Developments in Agent Communication*, volume 3396 of *LNAI*. Springer-Verlag, 2004.
- [11] Simon Parsons, Michael Wooldridge, and Leila Amgoud. An analysis of formal inter-agent dialogues. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, pages 394–401. ACM Press, 2002.
- [12] Anand S. Rao. AgentSpeak(L): BDI agents speak out in a logical computable language. In Rudy van Hoe, editor, *Seventh European Workshop on Modelling Autonomous Agents in a Multi-Agent World*, Eindhoven, The Netherlands, 1996.

- [13] Peter Rickard. *A History of the French Language*. Routledge (UK), 1989.
- [14] David Robertson. A lightweight coordination calculus for agent social norms. In *Declarative Agent Languages and Technologies*, New York, USA, 2004. a full day workshop occurring as part of AAMAS'04.
- [15] David Robertson. A lightweight method for coordination of agent oriented web services. In *Proceedings of AAAI Spring Symposium on Semantic Web Services*, California, USA, 2004.
- [16] David Robertson. Multi-agent coordination as distributed logic programming. In *Proceedings for International Conference on Logic Programming*, 2004.
- [17] Michael Rovatsos. *Computational Interaction Frames*. PhD thesis, Department of Informatics, Technical University of Munich, 2004.
- [18] John Searle. *(on) Searle on Communication*. Cambridge University Press, 1969.
- [19] John Searle. *Speech Acts*. Cambridge University Press, 1969.
- [20] Munindar P. Singh. A social semantics for agent communication languages. In Frank Dignum and Mark Greaves, editors, *Issues in Agent Communication*, pages 31–45. Springer-Verlag: Heidelberg, Germany, 2000.
- [21] Doug Walton and Eric C. W. Krabbe. *Commitment in Dialogue: Basic Concepts of Interpersonal Reasoning*. SUNY press, Albany, NY, USA, 1995.