Bridging the Specification Protocol Gap in Argumentation

Ashwag Maghraby, Dave Robertson, Adela Grando, and Michael Rovatsosr

School of Informatics, The University of Edinburgh, Edinburgh EH8 9LE, UK
A.O.Maghraby@sms.ed.ac.uk

Coordinating agents in open environments is a difficult problem that has engaged mul-ti-agent systems researchers on three broad fronts: argumentation (which focuses on the basis for negotiation between agents); electronic institutions (which focuses on the norms of social interaction in agent groups) and protocols (which focuses on deployment of interactions). None of these areas subsumes the others but there is a strong interaction between them in many cases. For example, if one wishes to construct a multi-agent trading system then this will contain negotiation, restrictions on group behaviour and protocols relevant to each agent. This is broadly analogous to the relationships between different views of formal system requirements in software engineering. Accordingly, the interesting challenge here is how (or whether) the specification of one of these components can be used to constrain the specification of the others. One way of addressing this issue is through the automated synthesis method, so the specific question that we ask is whether a generic argumentation representation (acting as a high level specification language) can be used to automate the synthesis of executable specifications in a protocol language capable of expressing a class of multi-agent social norms. As our argumentation language we have chosen the Argument Interchange Format (AIF is a generic specification language for argument structure). As our protocol language we have chosen the Lightweight Coordination Calculus (LCC is an executable specification language used for coordinating agents in open systems).

Fully automated synthesis starting only from the AIF is not possible because AIF is an abstract language that does not capture some concepts that are related to the interchange of arguments between agents (e.g. sequence of argument, locutions and pre- and post-conditions for each argument). An example of this problem occurs in one of the basic dialogue games stereotypes: ***A1*** and ***A2*** are reasoning about whether a particular penguin, Tweety, can fly: ***A1)*** Tweety flies. (making a claim); ***A2)***Why does Tweety fly? (asking for grounds for a claim); ***A1)*** Tweety is a bird, birds generally fly. (arguing: offering grounds for a claim); ***A2)*** Tweety does not fly because Tweety is a penguin, penguins do not fly. (stating a counterargument); ***A1)***You are right. (conceding an argument).

In this dialogue game, each agent responds in turn to the argument made by other agent. This flow of the dialogue is not captured by AIF (e.g. AIF does not record that a given argument has been made in response to an earlier argument). AIF only captures argument structures (e.g. it connect "Tweety flies" with its premises "Tweety is a bird, birds generally fly").

To remedy this problem, we extended the AIF diagrammatic notation to give a new, intermediate recursive visual high level language between the AIF and LCC called a Dialogue Interaction Diagram (DID). DID is an extension of AIF which provides mechanisms to represent, in an abstract way, the dialogue game protocol rules by giving an overview of the permitted moves (messages) and their relationship to each other. It restricts agent moves to unique-moves *(agents can make just one move before*

the turn-taking shifts and agents can reply just once to the other agents move) and immediate-reply moves *(the turn-taking between agents switches after each move and each agent must reply to the move of the previous agent)*. This restriction is quite strict but it still allows us to include a large class of argumentation systems in our synthesizer, for instance all argumentation systems that can be described as dialogue games. In general, we can synthesise arguments that can be described as a sequence of recursive steps (each of which involves turn-taking between the pair of agents) terminating in a base case. Given the turn-taking assumption, we can synthesise LCC protocols (which are executable) directly from DID specifications. However, a DID cannot explain how two or more agents can cooperate and interact with each other in situations where more complex protocols involving more than turn-taking are required.

To overcome this problem, we supply LCC-Argument patterns, which are re-usable, parameterisable LCC specifications that can be embedded in automated synthesis tools and used with DID to support agent protocol development. This allows us to reduce the effort of building more complex argumentation protocols by re-using design patterns repeatedly to generate argumentation protocols. The set of these more complex design patterns is, in theory, unbounded (for the same reason that design patterns in traditional software engineering are unbounded) but in practice families of interaction patterns occur. We have focused on those involving more than two agents where synthesized LCC protocols specify broadcasting methods to divide agents into groups composed of two agents (with these two-agent dialogues then being specified using DID).

Because, design patterns introduce greater scope for inaccuracy and error in the synthesis process (since a poorly designed interaction pattern may result in an inappropriate LCC protocol even with a perfect synthesis mechanism), we also provide a verification methodology based on model checking to verify the semantics of the DID specification used as a starting point against the semantics of the synthesised LCC protocol. The model checking system is built in three stages: (1) automatically mapping the LCC specification into an equivalent Coloured Petri Net (CPN). The formal semantics of the CPN model allows us to prove that certain (un)desirable properties are (un)satisfied in a LCC protocol. Proof of properties in LCC protocols mapped into CPNs is supported by a state-space technique, which is used to compute exhaustively all possible execution states; (2) automatically generating DID properties as a Standard ML(Meta-Language) specification; (3) automatically verifying the satisfaction of the Standard ML specification in the state-space graph computed from the LCC protocol.

Although the resulting synthesis and verification system is not an industry-strength specification tool, it demonstrates how automated synthesis methods can connect argumentation to the class of electronic institutions that can be expressed as protocols in a process language. This, potentially, could allow developers of argumentation systems to use specification languages to which they are accustomed (in our case AIF/DID) to generate systems capable of direct deployment on open infrastructures (in our case LCC).