

On the Role of Protocols in an Argument Interchange Format

Dave Robertson, Jarred McGinnis, Chris Walton

Informatics
University of Edinburgh

Abstract

In this note we take the view that our community is unlikely to agree, in detail, on a single semantics or ontology of forms of argument. We therefore favour the definition of a core standard that fixes a few standard operators concerning the protocol for interaction during an argument, leaving the details of specific forms of argument to personal choice of developers who extend the core standard. We use as an example the LCC protocol language, although our arguments would apply to any similar language.

In cases where there is substantial experience of using the same system then it is possible to construct comprehensive standards of language usage. This is the case for programming language standards (such as Prolog, dialects of ADA, *etc*). By contrast, in areas where standardisation of more abstract concepts is required it appears to be much harder to achieve consensus (because abstract concepts are difficult to pin down uniquely in a simple way). In this circumstance it often is expedient to define precisely a core standard, containing only those elements essential to getting the job done, and then allow extensions to this core in a controlled (but perhaps less precise) way. An example of this form of standardisation is the Process Interchange Format (PIF) which is a standard for describing processes. The PIF core contains a small number of very generic concepts at the heart of that standard and then allows those with specific process description needs to meet their own requirements by building on that core.

What is core for argument interchange? One way to approach this question is to start from the “sharp end” of computation. We do not, however, want to limit a standard to a particular implementation language so an appropriate level might be that of interaction protocols. The practical advantages of including a protocol language in a core standard are:

- If the protocol can be used for computation then the standard is, effectively, a programming standard and history suggests that such standards tend to be durable because they connect to practise (or fail to connect and then die cleanly).
- If the protocol is also declarative - hence independent of current fashion in low level implementation languages or basic communications protocols - then it can support formal analysis and verification more readily.
- There is a natural (for software engineers) notion of pattern in the design of protocols and this is one approach to extension from a core protocol syntax to a (more interesting) set of extensions via patterns.

Protocols are an area where traditional computer science helps supply standards. For example, Figure 1 defines the syntax of the Lightweight Coordination Calculus (LCC) that uses a combination of traditional specification drawn from CCS and logic programming. For details of LCC see, for example [2]. An interaction model in LCC is a set of clauses, each of which defines how a role in the interaction must be performed. Roles are described by the type of role and an identifier for the individual agent undertaking that role. The definition of performance of a role is constructed using combinations of the sequence operator (*‘then’*) or choice operator (*‘or’*) to connect messages and changes of role. Messages are either outgoing to another agent in a given role (*‘⇒’*) or incoming from another agent in a given role (*‘⇐’*). Message input/output or change of role can be governed by a constraint defined using the normal logical operators for conjunction, disjunction and negation. Notice that there is no commitment to the system of logic through which constraints are solved - on the contrary we would expect different agents to operate different constraint solvers. Hence the standardisation in LCC is on the generic language for describing interaction (only) and in this sense it is “core”.

To demonstrate how LCC applies to argumentation, we give in Figure 3 a protocol for the Information-seeking dialogue type described on page 12 of [1] (repeated in Figure 2 for convenience). The details of the example are not important to this informal note. What matters is that we are standardising on a style of description that is close to computation - in this case quite close to logic programming (despite the process operators) where we already have a successful ISO standard.

Our choice of LCC is, of course, because it is close to the authors’ own experience. Nevertheless, it has a specific advantage as a protocol standard for automated argument because it is oriented to declarative description of the interaction independent of the agents concerned, and is supported by mechanisms allowing portability of the

$Model$:= $\{Clause, \dots\}$
 $Clause$:= $Role :: Def$
 $Role$:= $a(Type, Id)$
 Def := $Role \mid Message \mid Def \text{ then } Def \mid Def \text{ or } Def$
 $Message$:= $M \Rightarrow Role \mid M \Rightarrow Role \leftarrow C \mid M \Leftarrow Role \mid C \leftarrow M \Leftarrow Role$
 C := $Constant \mid P(Term, \dots) \mid \neg C \mid C \wedge C \mid C \vee C$
 $Type$:= $Term$
 Id := $Constant \mid Variable$
 M := $Term$
 $Term$:= $Constant \mid Variable \mid P(Term, \dots)$
 $Constant$:= lower case character sequence or number
 $Variable$:= upper case character sequence or number

Figure 1: LCC syntax

An information seeking dialogue between an information seeking agent, A and another agent B about proposition, p , is as follows:

1. A asks $question(P)$.
2. B replies with either $assert(p)$ or $assert(\neg p)$ if it can, and $assert(U)$ if it cannot. U indicates that, for whatever reason, B cannot give an answer.
3. A either accepts B 's response or $challenges$. U cannot be challenged and as soon as it is asserted the dialogue terminates without the question being resolved.
4. B replies to a challenge with an $assert(S)$, where S is the support of an argument for the last proposition challenged by A .
5. Go to 3 for each proposition in S in turn.
6. A accepts p if its acceptance attitude allows.

Figure 2: Information seeking dialogue from [1]

$a(\text{information_seeker}, A) ::$
 $\text{question}(P) \Rightarrow a(\text{information_supplier}, B) \text{ then}$
 $\text{assert}(R) \Leftarrow a(\text{information_supplier}, B) \text{ then}$
 $a(\text{info_negotiator}([R], B), A)$

$a(\text{info_negotiator}(S, B), A) ::$
 $\left(\begin{array}{l} a(\text{proposition_negotiator}(P, B), A) \leftarrow \text{select}(P, S, Sr) \text{ then} \\ a(\text{info_negotiator}(Sr, B), A) \end{array} \right) \text{ or}$
 $\text{null} \leftarrow S = []$

$a(\text{proposition_negotiator}(P, B), A) ::$
 $\text{accept}(R) \Rightarrow a(\text{info_responder}, B) \leftarrow \text{not}(R = \text{unknown}(P)) \wedge \text{acceptable}(P) \text{ or}$
 $\left(\begin{array}{l} \text{challenge}(R) \Rightarrow a(\text{info_responder}, B) \leftarrow \text{not}(R = \text{unknown}(P)) \wedge \text{not}(\text{acceptable}(P)) \text{ then} \\ \text{assert}(S) \Leftarrow a(\text{info_responder}, B) \text{ then} \\ a(\text{info_negotiator}(S, B), A) \leftarrow \text{support_list}(S) \end{array} \right) \text{ or}$
 $\text{null} \leftarrow R = \text{unknown}(P)$

$a(\text{information_supplier}, B) ::$
 $\text{question}(P) \Leftarrow a(\text{information_seeker}, A) \text{ then}$
 $\left(\begin{array}{l} \text{assert}(\text{unknown}(P)) \Rightarrow a(\text{information_seeker}, A) \leftarrow \text{is_unknown}(P) \text{ or} \\ \left(\begin{array}{l} \text{assert}(P) \Rightarrow a(\text{information_seeker}, A) \leftarrow \text{believed}(P) \text{ then} \\ a(\text{info_responder}, B) \end{array} \right) \text{ or} \\ \left(\begin{array}{l} \text{assert}(\text{not}(P)) \Rightarrow a(\text{information_seeker}, A) \leftarrow \text{not_believed}(P) \text{ then} \\ a(\text{info_responder}, B) \end{array} \right) \end{array} \right)$

$a(\text{info_responder}, B) ::$
 $\text{accept}(R) \Leftarrow a(\text{proposition_negotiator}(P, B), A) \text{ or}$
 $\left(\begin{array}{l} \text{challenge}(R) \Leftarrow a(\text{proposition_negotiator}(P, B), A) \text{ then} \\ \text{assert}(S) \Rightarrow a(\text{proposition_negotiator}(P, B), A) \leftarrow \text{support}(R, S) \text{ then} \\ a(\text{info_responder}, B) \end{array} \right)$

Figure 3: LCC protocol for an information seeking dialogue from Figure 2

protocol between agents. Thus a standard for argument developed by one agent may directly be communicated to another in a simple and precise way.

Argumentation is, however, more than just use of protocols. One important issue is the categorisation of forms of argument (attack, defence, *etc*). If we wish to standardise what these mean in automation then there must be some computational frame of reference for them. One way to supply this is through properties of protocols. For example, an attacking argument might be one in which an assertion is made that contradicts an assertion by another agent. For some protocols, this property might be demonstrated once and for all. For others it may only be evident at run time. Either way, it can be described as a property, whether checked or not.

As a summary of the argument presented above - we propose as a basic engineering approach to standardisation the following:

A core protocol language standard : LCC is an example of a lightweight protocol language oriented to declarative programming.

An extensible system of patterns : used to describe standard practise in argumentation domains. Figure 3 is an example of a pattern, although it is not clear whether patterns should be expressed directly in a core protocol language or in some extension of it that provides additional meta-level language constructs.

An extensible set of argument properties : used by engineers to categorise argument protocols and control their design and use.

Acknowledgements

Thanks to Iyad Rahwan for supplying the germ of the idea for our note with his position statement to this meeting.

References

- [1] S. Parsons, M. Wooldridge, and L. Amgoud. Properties and complexity of formal inter-agent dialogues. *Journal of Logic and Computation*, 13(3), 2003.
- [2] D. Robertson. Multi-agent coordination as distributed logic programming. In *International Conference on Logic Programming*, Sant-Malo, France, 2004.